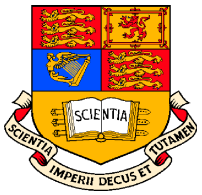
PARALLEL ANISOTROPIC UNSTRUCTURED MESH OPTIMISATION AND ITS APPLICATIONS

by

Gerard Gorman, BSc, MSc

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy and the Diploma of Imperial College



Computational Physics and Geophysics Group
Department of Earth Sciences and Engineering
Imperial College of Science, Technology and Medicine
University of London

March, 2006

To my parents, Dolores and Eugene

Reasonable people adapt themselves to the world. Unreasonable people attempt to adapt the world to themselves. All progress, therefore, depends on unreasonable people.

George Bernard Shaw

Abstract

This thesis is primarily concerned with the parallelisation of a mesh adaptivity method, used in unstructured mesh finite element modelling, and the dynamic load-balancing of subsequent computation for distributed memory parallel computers. The key features of the mesh adaptivity method are: it is an optimisation based method; the optimisation seeks to minimise an objective function written in terms of a Hessian based metric tensor; optimisations are carried out iteratively through trials of local changes in the mesh connectivity (h-method) and node movement (r-method). The Gauss-Seidel iterative nature of the algorithm complicates the optimisation of the elements that are shared between sub-domains. The approach adopted in this work locks these shared regions so that the serial algorithm can be applied to each sub-domain independently. Careful tuning of the subsequent graph partitioning ensures that substandard elements that were previously located are later made internal to a sub-domain so that they are free to be optimised when the serial adaptivity algorithm is reapplied. The additional cost of the required graph repartitioning and data remapping can be effectively hidden by the requirements and cost of dynamic load-balancing. A range of computational fluid dynamics examples, which use the described method within a finite element method, are presented. It is found that graph repartitioning and data migration scales well with the number of processors and the total cost of these operations is far less than that of serial mesh optimisation. Load-imbalances may occur during parallel mesh optimisation for problems which require substantial mesh optimisation in just a few localised regions. This however is not found to be a problem as the total cost of mesh optimisation is, for most practical applications, much less than the cost of the solver. A novel application of the mesh adaptivity to shape approximation optimisation for bathymetry data is also presented. The thesis concludes with a discussion of the advantages and disadvantages of the current method as well as future work to be undertaken.

Acknowledgements

I would like to thank everyone who contributed to this work. In particular I would like to thank my supervisors Dr. Christopher C. Pain, Dr Cassiano R. E. de Oliveira and Professor Antony J. H. Goddard who provided constant tuition, guidance and sponsorship. I was also lucky to have been able learn a great deal from constant discussion with colleagues involved in related work — in particular Colin Cotter, Matthew Eaton, Matthew Piggott, Philip Power and Adrian Umpleby, all of whom made valuable intellectual contributions to this work. I would also like to thank my good friends at the Scientific Computing Group at NUI, Galway for always letting me syphon off spare (or hog) CPU cycles on their machines.

I would like to thank my parents - who I cannot thank enough - and the rest of my family for the unlimited support and patience over the years. I would like to thank my wife, Maria Nora, whom I was very lucky to meet at the outset of this work and provided unwavering support - and of course our son Ian for providing the final deadline for this work!

Contents

[1] An updated version of this work is available: Gorman et al. [55]

List of Figures

Nomenclature

Abbreviations

EEList	Element-Element adjacency List
EEList$_i$	list of elements adjacent to element i
EEList$_{ij}$	j^{th} element adjacent to element i
ENList	Element-Node adjacency List
ENList$_i$	list of nodes in element i
ENList$_{ij}$	j^{th} node in element i
EgEList	Edge-Element adjacency List
EgEList$_i$	set of elements adjacent to edge i, no duplicates
EgEList$_{ij}$	j^{th} element adjacent to edge i
NEList	Node-Element adjacency List
NEList$_i$	set of elements that contain node i, no duplicates
NEList$_{ij}$	j^{th} element that contains node i
NNList	Node-Node adjacency List
NNList$_i$	set of nodes adjacent to node i, no duplicates
NNList$_{ij}$	j^{th} node adjacent to node i
Tetra4	4 node linear tetrahedral element
Tetra10	10 node quadratic tetrahedral element

Acronyms

AMCG	Applied Modelling and Computation Group
AMR	Adaptive Mesh Refinement
CFL	Courant-Friedrichs-Lewy
DDM	Domain Decomposition Methods
FEM	Finite Element Method
FVM	Finite Volume Method
GNN	Global Node Number
ICOM	Imperial College Ocean Model

LBB Ladyzhenskaya-Babuska-Brezzi stability condition

LES Large Eddy Simulation

LUT Lookup Table

LNN Local Node Number

MNO Minimum Node Owner (MPI rank number) of an edge or element

MPI Message Passing Interface

SIMD Single Instruction Multiple Data

SPMD Single Program Multiple Data

UNN Universal Node Number

Latin Characters

$\boldsymbol{h}_{ij}$ the vector of node numbers corresponding to solution variables that need to be passed from process i to process j

H Hessian

$\boldsymbol{I}$ the identity matrix

V_e volume of an element e in Euclidean space

$\tilde{V}_e$ volume of an element e in metric space

Calligraphic Characters

$\mathcal{D}_s$ the set of elements which contain nodes assigned to domain s

$\mathcal{E}_e$ the set of elements that share a node with element e

$\mathcal{F}$ functional used to gauge the quality of the mesh

$\mathcal{I}_e$ the set of nodes at the vertices of element e.

$\mathcal{L}_e$ is the set of edges of element e

$\mathcal{N}_l$ the set of nodes in edge l

$\mathcal{R}$ maps a partition wide numbering, GNN, to unique number for the whole domain, UNN

Mathematical Symbols

$\|x\|_\infty$	L^∞-norm of a vector x
$\|x\|_2$	L^2-norm of a vector x
$\|x\|$	number of elements in vector x
$\lceil x \rceil$	maximum value in vector x
$\lfloor x \rfloor$	minimum value in vector x
$\#x$	cardinal number of set x
(x)	the list/vector of x
$\{x\}$	the set of x
$x \leftarrow a$	assigns a to x
$(x) \leftarrow a$	appends entity a to the list/vector x
$\{x\} \leftarrow a$	inserts entity a into the set $\{x\}$
$x[i]$ or x_i	indexes the i^{th} entity of list/vector x
$\Re$	the set of real numbers

Chapter 1

Introduction

Contents

This work is concerned with the development of a parallel unstructured mesh optimisation method and supporting technology. While many of the methods described here are of general applicability to mesh based FEM/FVM computation (throughout this thesis, *mesh* should be understood to mean *unstructured mesh*), the focus is on 3D models that use simplex meshes which are unstructured.

This chapter begins by outlining the motivation for using unstructured mesh optimisation methods. This is followed by a review of optimal error control which is central to the notion of optimisation. This is followed by a brief description of current trends in mesh adaptivity. Issues relating to the parallel implementation of each method will also be discussed. The chapter will conclude with an outline of the rest of the thesis.

1.1 Background

Unstructured mesh based computation will form the basis of the next generation of models to be developed over the next decade for ocean and atmospheric modelling [1], multiphase flows, and radiation transport. The most commonly cited motivation for the use of unstructured meshes, as juxtaposed to structured meshes, is that they are highly suited to modelling with complex geometries (domain boundaries), while the primary disadvantage is generally seen as the significant computational and storage overheads incurred. Because of this, the potential of such models are only fully realised with the use of mesh adaptivity techniques [2]. Adaptive mesh methods, which allow mesh resolution to be focused on where it is needed, give rise to a clear potential for unstructured meshes to be more efficient than structured meshes in terms of computational cost and storage.

Critical to any state-of-the-art unstructured mesh fluids model is the ability to dynamically resolve developing solution features (e.g. shock waves, fronts, eddies) whose positions are not necessarily known a priori to the simulation. Mesh adaptivity methods allow the mesh to be locally optimised to resolve such flows in response to an error metric which is generally derived from either the solution, or the partial differential equations describing the system dynamics and modelling goals [3, 4, 5]. In addition to this, the error metric (described in detail in the next section) can be modified to reflect auxiliary information. An example of this would be an ocean circulation model in the Mediterranean where the subject of investigation is dynamics close to Malta. In cases such as this, it is natural to define an *importance* function within the domain which is unity close to Malta and decays with distance from the shelf surrounding Malta. This function can then be used to rescale the desired mesh length scales as defined by an error metric.

Mesh optimisation methods locally adapt the model resolution to resolve solution features under the guidance of some error metric derived from either the solution (such as the interpolation error in the piecewise polynomial representation), or the partial differential equations describing the system dynamics (*a posteriori* errors) and modelling goals. There are three broad categories of adaptive methods: the *h-method*, which refines, coarsens and modifies the connectivity of elements locally; the *p-method*, which adjusts locally or globally the order of the polynomial expansion in the element basis functions;

the *r-method*, which relocates a fixed number of grid points to regions where high resolution is needed. The naming and definitions of the adaptive methods adopted here are based on those used by George [6], but in the literature these categories are also known as h,r,p-version, refinement or adaptivity. AMR (the acronym is generally taken to mean adaptive mesh refinement as applied to structured meshes) applies the h-method through local grid subdivision. This method is popular as it is straightforward to implement and can be used for structured meshes. However, the method results in hanging nodes that must be constrained to avoid numerical problems (e.g. spurious wave reflections), thus making the calculations more involved ([7], page 402). A similar argument can be made regarding mesh-nesting methods. A simple solution to the hanging node problem is to extend the element subdivision to neighbouring nodes until a domain boundary is reached. However it is clear that this results in unnecessarily large numbers of degrees of freedom being added to some regions of the domain. This is in contrast to the h-method used with unstructured simplex meshes which, as will be seen in the next chapter, only perform local mesh modifications and always result in a conformal mesh with no hanging nodes.

As modelling interest is increasingly in high fidelity solutions for complex problems, parallel computers, and complementary parallel methods, are generally adapted to provide the computational power required. Indeed, the demand for model realism has always outpaced developments in computer technology and computational methods, and the trend is set to continue. Principal challenges associated with developing a parallel mesh adaptivity method include consistency of shared node and element information between subdomains, minimisation of communication overhead costs, and dynamic load-balancing. Mesh data consistency refers to the requirement that all processors that have information regarding particular mesh entities (e.g. nodes, elements, auxiliary), have that information correctly updated as part of the mesh adaptivity process and that the complete mesh remains conformal after the application of mesh optimisation (i.e. no hanging nodes are allowed). After mesh optimisation has been performed, load-imbalances are to be expected as nodes and elements are in general added and deleted according to solution requirements and thus the distribution of work may vary across subdomains. The obvious exception to this is the r-method as the number of nodes and mesh connectivity does not change — clearly an attractive property for parallel computation. However, it has been

found to be impractical to use pure r-methods ([7], pages 402–403) and the best results are generally obtained by combining the r-method with the h-method [8]. For these reasons there is a great deal of interest in dynamic load-balancing and cost modelling for parallel mesh optimisation — deciding when the mesh needs to be repartitioned among the processors involved in the computation, methods for calculating such a repartitioning and efficiently realising the new partitioning through data migration (in the literature this is also referred to as data remapping).

1.2 Error control

Consider the 1D function illustrated in Figure 1.1. It is clear that in order to faithfully represent this function using piecewise linear splines (or other low order polynomial splines), more splines are necessary within the shaded regions than outside. It is this simple observation that motivates mesh adaptivity methods. The overall mesh adaptivity procedure can be viewed as comprising two phases. The first phase involves evaluating solution errors. This may involve estimating in some sense the discrepancy, or potential discrepancy in the case of sensitivity estimates, between the computed solution and the real solution for example. Perhaps the most readily available error measure is the interpolation error (an excellent introduction to this and the theory of error control is given by Simpson [9] and D'Azevedo et al. [10]). In 2D, consider a triangular element, T, and a linear interpolant $p(x)$ of a convex quadratic function $f(x)$ on the vertices of T. The error at any point in T is simply

$$e_T(x) = f(x) - p(x).$$

This error is also a quadratic function and has the same Hessian, H, as f (the second derivatives of p are zero as it is piecewise linear). As will be shown in Section 2.4, if the eigenvalues of the Hessian are modified so that $\lambda_i \leftarrow abs(\lambda_i), \forall i$, then the contour lines of constant error form concentric ellipses where the ellipse corresponding to an interpolation error of zero passes through the vertices of T and increases toward the centre of the ellipses, as illustrated in Figure 1.2. Note that this modification to the Hessian is justified as it is the magnitude of the curvature of the solution that is significant in this

context and not its sign. The maximum error in T is then either the value at the centre of the ellipses, in the case where the centre is located within T, or on the boundary of T at the midpoint of the edge closest to this centre [10]. Clearly then any triangle which has all three vertices on the same error contour will have the same maximum error if the centre of the ellipses lies inside the triangle. At the simplest level, the send phase of mesh adaptivity then tries to ensure that this maximum error is within some error band, $\epsilon \pm \Delta\epsilon$, by adjusting the size of the element. Mesh adaptivity becomes an optimisation problem (thus qualified term, mesh optimisation) when the maximum efficiency mesh is sought; a maximum efficiency mesh is formed when each element has a maximum specified error of ϵ and the element of maximum volume is formed within the zero error contour for each T [9]. Variants of this problem can use many different error measures such as gradient errors and other solution based error measures [10, 11] or a posteriori errors [12, 13, 14]). These error measures are typically combined with other modelling constraints, such as geometric constraints or constraints on the maximum number of degrees of freedom that can be solved for. The final metric is typically either a scalar, which is used to determine elements to be refined or de-refined, or a metric tensor.

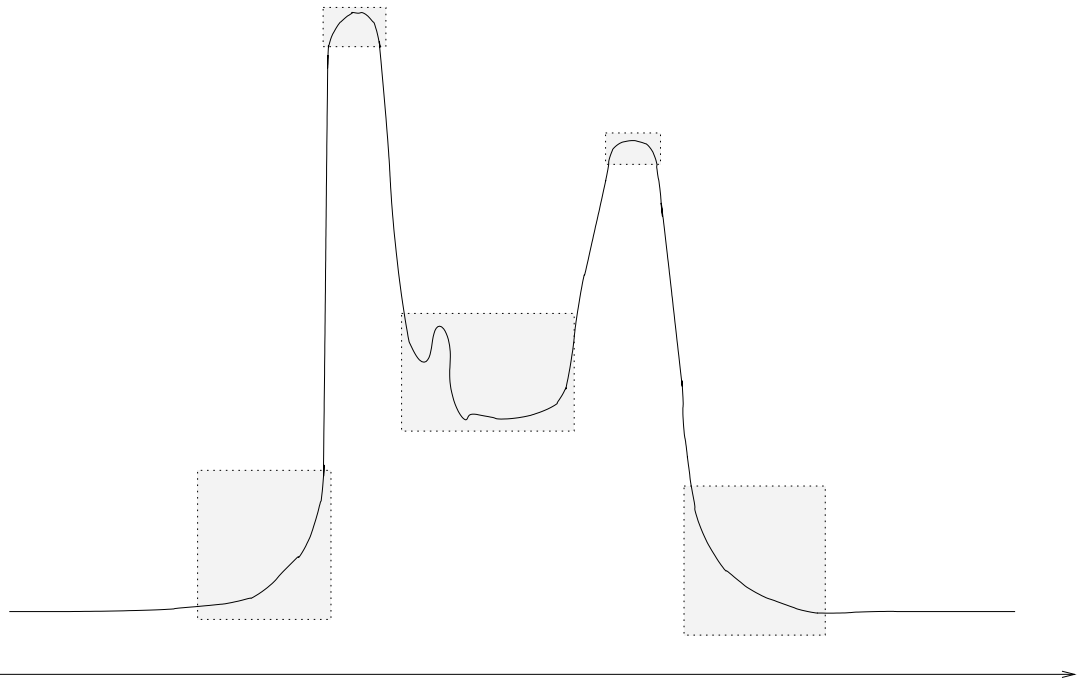


Figure 1.1: An adaptive mesh method in 1D would include a higher density of nodes in the shaded regions than elsewhere in the domain in order to capture the more rapid variation of the function.

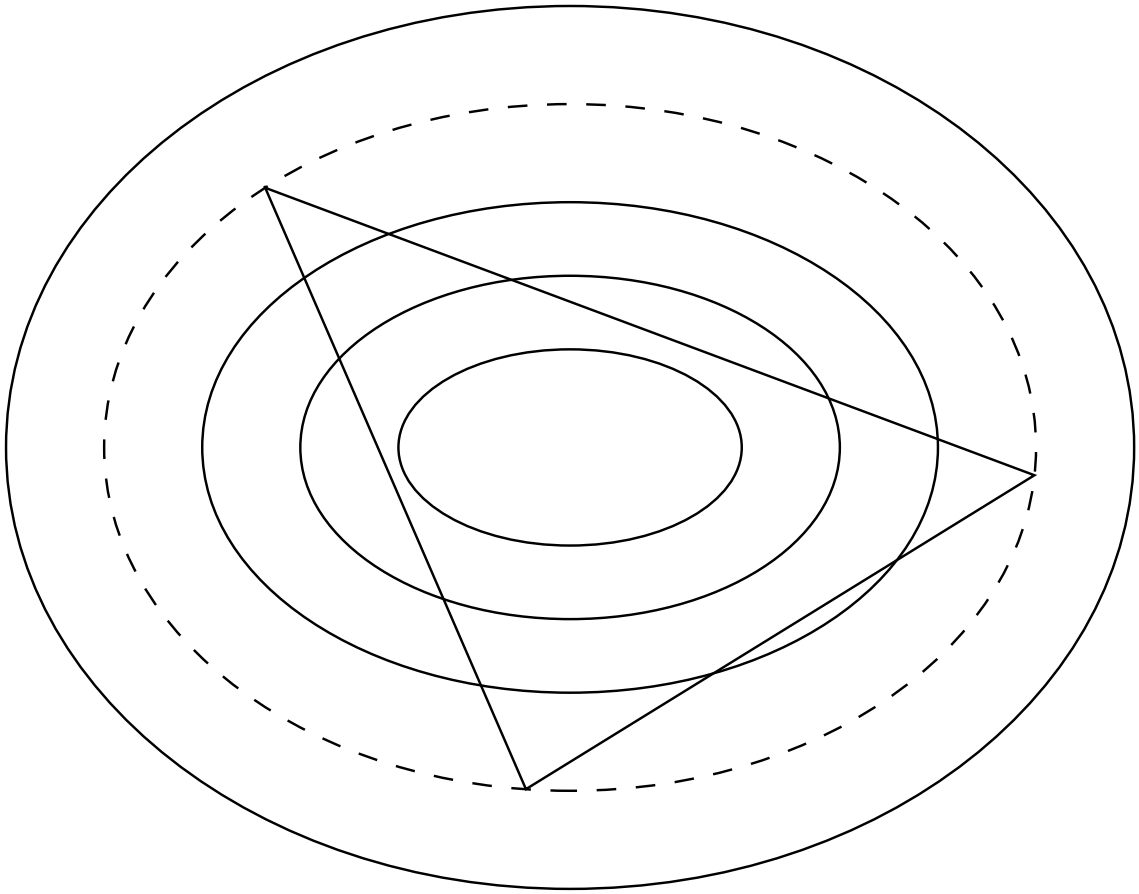


Figure 1.2: Contours of interpolation error on a triangular element. The dashed contour is the contour of zero error. Notice that in general the triangle of maximum area, that is circumscribed by any contour, will in general be anisotropic.

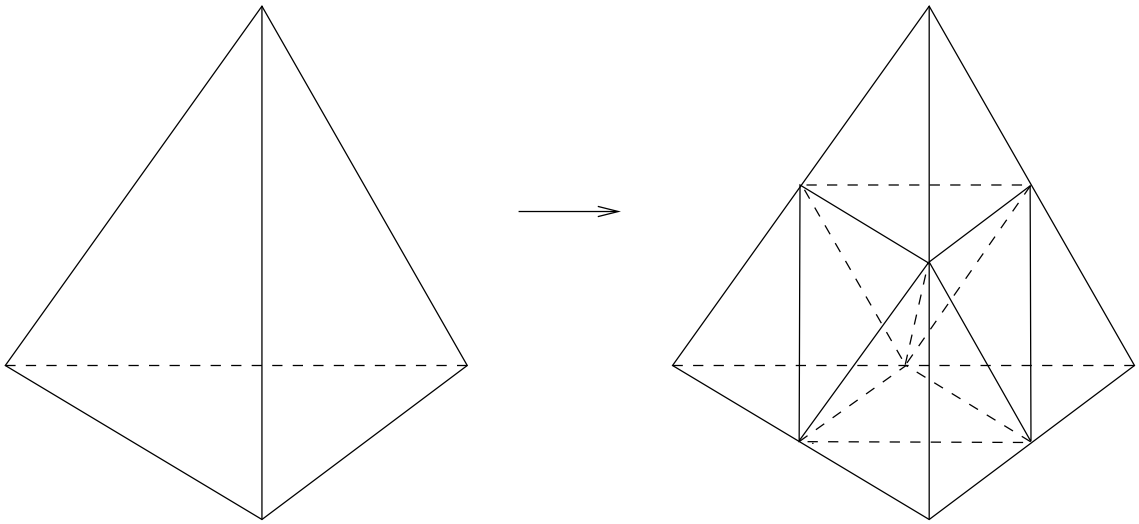
1.3 Unstructured mesh adaptivity and optimisation

Given an error metric, the question becomes how to obtain a more efficient mesh - how to generate a mesh whose error is close to some user specified value (for the sake of the present discussion it is assumed that any other mesh constraints are included in this error). As previously stated mesh optimisation methods can be categorised as being r-method, h-method, p-method or some combination of these. The p-method will not be considered in this work other than noting that when it is applied globally there are very few parallel computing issues and there is much to recommend it. All of these techniques focus on modifying an existing mesh to fulfil some objective function. An alternative is to regenerate an entirely new mesh, using standard mesh generation techniques such as

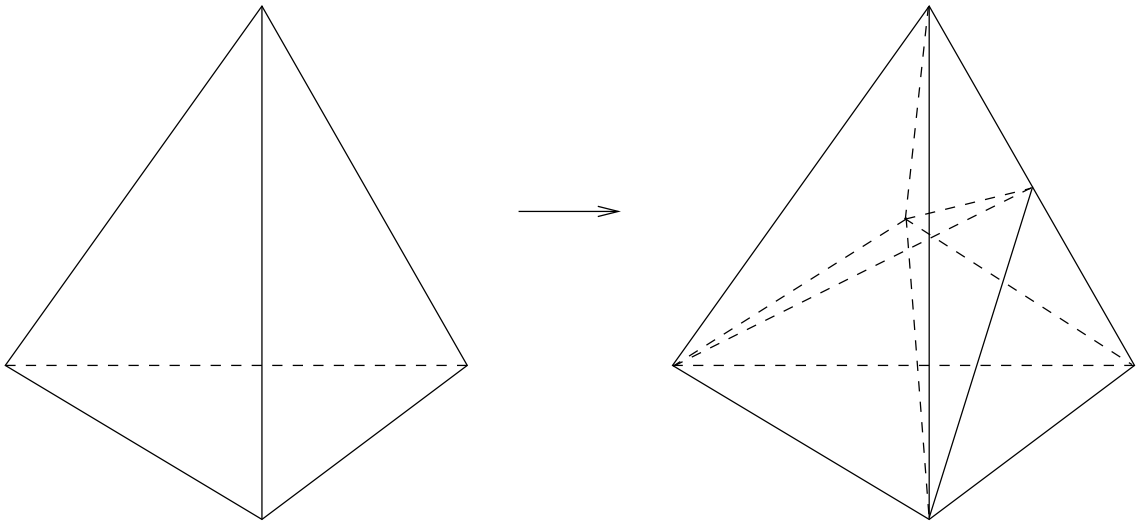
advancing front or Delaunay triangulation. In this case the mesh generation uses the error metric to ensure that the correct density of nodes are inserted in the mesh [15, 16, 17, 18, 19, 20]. However this is in general expensive as compared with local optimisation based adaptive methods (consider for example if only a small region in a large domain requires modification or if a transient problem is being solved).

Hierarchical based mesh adaptivity methods form a widely, and very successfully, employed type of h-method [21, 22, 23, 24, 25, 26, 27, 28]. The method operates by inserting nodes into the edges to be refined, and subdividing the *parent* elements surrounding that edge into several smaller *child* elements. Once a systematic data structure has been formed at the outset of the solution process which stores parent-child relation information, refinement and de-refinement is fast and efficient. The method also has the advantage that it can be used to form the basis of a multigrid method [22, 25]. For tetrahedral elements a good explanation of the algorithm is given by Speares et al. [28]. Once mesh edges have been marked for refinement, elements can be grouped into two classes: elements which contain 4 edges or more that must be refined and are subdivided using *regular subdivision* (see Figure 1.3 (a)); elements that have 1 to 3 edges to be refined are treated with *green subdivision* (see Figure 1.3 (b)). In the case of regular subdivision, when the inner octahedron is being triangulated it is usual that the longest inner diagonal forms the central edge [26] although there are other possibilities [27]. Elements which require 1-3 edges to be refined are referred to as green elements and need to be subdivided in such a way as to regain mesh conformity. This can be done by inserting an extra node into the interior of the element (usually at the centroid) and connecting this node to the vertices of the element and to the middle nodes of the bisected elements. Some elements formed in this way may be of poor quality thus adaptivity is constrained such that green elements cannot be further refined. If a green element requires refinement, it is first coarsened and the parent is then refined using regular subdivision. It is shown by Ong [27] that the degradation in mesh quality due to this method remains bounded. Other variations of green subdivision also exist that do not involve the insertion of an extra node (see [29]).

Scalable parallel algorithms for this adaptivity method have also been developed [30, 31, 32, 33, 34]. Generally the approach is to partition the mesh on the coarsest level of the mesh hierarchy. While this gives more coarse grained control of the load-balancing it



(a) Regular subdivision



(b) Green (1:6) subdivision

Figure 1.3: (a) All edges are bisected and the interior is triangulated; (b) A node is inserted into the centre of the element and connected to each of the vertices and any hanging nodes introduced by regular subdivision of neighbouring elements.

also greatly simplifies the problem of coarsening and refining mesh entities shared between processors as well as the complexity of the data structure that stores the element hierarchy. The key issue when adapting elements in parallel is that of dealing with green elements. When a green element is to be refined it is first coarsened and then a regular subdivision is performed on the parent element. This operation will bisect an edge that was not previously marked for refinement and thus elements sharing that element now must be subdivided to regain mesh conformity. This "knock-on" effect can clearly continue. When this is performed in parallel such knock-on effects will sometimes cross domain boundaries. This is handled by forming a list of all those edges that have been marked for refinement at the subdomain interface and communicating this list to neighbouring elements once all other adaptations have been made. This avoids the need for numerous small communications that would be detrimental to the method's efficiency — though it is noted that communication required to maintain data consistency is still significant [32]. The adaptive procedure is then reapplied until all elements marked for refinement have been refined and a conformal mesh is obtained. Another important issue is that of dynamic load-balancing. Usually after an adaptive step the load-balance is calculated across the processors (this can be calculated as being the ratio between the minimum number of nodes on any partition and the maximum). A new graph partitioning and data migration is then performed if this drops below some threshold and the cost of data migration justifies the cost of data migration. It is noted that this method appears to be particularly challenging to implement principally because of the complexity of the required data structures and cost of migrating these data structures between processors [33] — although this is difficult to quantify.

An alternative class of mesh adaptivity methods rely instead on local changes (or elemental changes) to the mesh connectivity. These elemental operations can be grouped into three basic categories: point insertion/deletion (e.g. [23, 24, 35]); local reconnection methods (e.g. [36, 37, 38, 39]); mesh smoothing, also called r-method (e.g. [40, 41, 42, 43]). Many practitioners have demonstrated that all three operations can be combined effectively (e.g. [8, 3, 4]). These methods are generally referred to as mesh optimisation methods as they involve the definition of some objective function (some notion of the quality of an element) and elemental modifications are performed on the

mesh in an attempt to reach some goal as defined by that objective function. In the case of mesh optimisation being applied prior to a solution being computed, objective functions are based solely on geometric criteria such as the Jacobian matrix (the determinant of which provides an effective objective function for untangling) or limiting small and large dihedral angles (see Knupp for an excellent discussion on geometric based objective functions [44]). However, when mesh optimisation is used as part of the solution process, a posteriori error measures or error norms derived from the solution (e.g. interpolation errors) must also be taken into account in the objective function. This is usually accomplished by defining a metric tensor (e.g. [9, 45, 3]) which allows local anisotropic features in a solution to be made isotropic through a coordinate transformation. As is shown in Chapter 2, the metric tensor can be defined so that the optimal simplex element (in terms of efficiency and error) is an equilateral element with edges of unit length. An optimisation problem can then be formulated by defining an objective function in terms of the metric tensor. Although the problem of finding the optimal mesh is suspected to be NP-complete, the methods described above robustly generate good quality meshes. In addition Ollivier-Gooch has recently developed an optimised edge contraction algorithm specifically for generating a hierarchy of progressively coarser meshes so that geometric multigrid solver methods can also be employed [46].

The use of a metric tensor in directing the shape of elements in a mesh can lead to anisotropic elements when the solution is itself anisotropic. Traditionally anisotropic elements have been considered to result in numerical algorithms with both increased finite element discretisation errors and poorly conditioned matrices. It can be shown however that this is not necessarily the case provided that large (close to π) angles are avoided[1], and the anisotropic elements are aligned well with the problem physics or solution, so as to obtain a locally isotropic interpolation error (see [4] and the references therein).

When the fore mentioned adaptive methods are applied in parallel they are possibly more involved than hierarchical methods, from a data consistency perspective, as they rely on Gauss-Seidel type sweeps of the entire mesh, and the mesh must of course be conformal after mesh optimisation has been applied. A colouring based algorithm is used by some

[1]Large angles result in element interpolation error contours that do not form concentric ellipses with the center contained within the element as in Figure 1.2

investigators to accomplish this ([47, 48, 49]). To ensure that the shared mesh entities are not updated simultaneously on different partitions, a task graph for each given elemental operation must first be defined. The task graph defines the mesh entities that must be updated when an elemental operation is performed and thus defines that data that must be synchronised at the end of an adaptive step. Let the entity to be operated on be the first vertex of the task graph. All mesh entities that need to be updated because of the operation define the remaining graph nodes. The edges of the task graph are all these vertices connected to the first vertex. For example, for node smoothing this is simply the node being moved and the elements that contain that node (see Freitag et al. [48] for examples of other task graphs). The graph formed by connecting these task graphs is then coloured (in practice maximum independent sets are calculated as an optimal graph colouring is not sought). Broadly speaking, the parallel mesh adaptivity is then carried out by sweeping through graph entities of the same colour in sequence, synchronising changes after each colour has been processed (for full details see [49]). However this algorithm is complex to implement and would necessitate a significant rewrite of an existing serial mesh optimisation code. In addition, it is not clear how significant the overall cost of the synchronisation steps might be on distributed memory parallel computers.

However there is an alternative approach. Any of the mesh adaptivity methods discussed also induce load-imbalances due to the fact that the number of degrees of freedom varies. This requires an additional dynamic load-balancing step after mesh optimisation has been applied (or between a coarsening and refinement phases of mesh adaptation). The approach taken in this work is to overlap the usual communication overhead associated with data consistency when adapting a mesh in parallel with the communication cost of dynamic load-balancing.

1.4 Outline of thesis

The remainder of this thesis is outlined as follows. The next chapter describes the serial mesh optimisation method used in this work. The basic method is that originally developed by Pain et al. [3] but several significant innovations have been contributed. These

innovations include geometry constraints for discrete surfaces, metric tensor gradation control and adaptive time stepping. Chapter 3 describes the parallel implementation of the mesh optimisation method and dynamic load-balancing. The method relies on the novel use of edge weights, defined in terms of the error metric, with a graph partitioner to discourage mesh regions deemed more suboptimal from being partitioned. This minimises the number of shared elements that require adaption. This is combined with a fast, efficient data migration algorithm which, due to its specialised nature, minimises the overhead of parallel mesh optimisation and dynamic load-balancing. Chapter 4 deals with the construction and maintenance of halos for different discretisations. This chapter provides useful technical details of the overall algorithm operating with the DDM within the FEM codes considered. Chapter 5 focuses on a novel application of mesh optimisation methods to shape approximation optimisation for bathymetry datasets. This method facilitates the practical application of mesh optimisation to ocean modelling where the geometry must not be modified but must also be accurate while not overly constraining the solution mesh. Simulations results and empirical scalability results are presented in Chapter 6. The final chapter draws conclusions from the work presented and discusses possible future directions of mesh optimisation research.

Chapter 2

Mesh optimisation

Contents

2.1 Introduction

A key difficulty in the use of static computational meshes is that the mesh is generated a priori to the solution. Traditionally this has meant that numerical practitioners have had little control over the errors associated with the solution of a particular problem, at best aiming to limit the maximum error occurring in the domain by using a sufficiently fine mesh. This makes it difficult to ensure that the computational mesh has sufficient resolution to resolve the phenomena of interest, especially as the solution evolves for time-dependent problems, although it is also true for time-independent problems. Due to practical limitations on computer memory and time required to compute a solution, creating a sufficiently fine mesh everywhere in the domain is generally not an option. There is also a question of computational efficiency. Whereas in some areas the solution may be varying rapidly (i.e. have a high curvature associated with the solution variables), in other areas the solution may be relatively uniform and thus a high mesh resolution is not necessary for the solution to converge and the solution features of interest to be captured.

Tetrahedral mesh optimisation (or mesh adaptivity) provides a method for locally modifying a tetrahedral mesh in order to maintain solution errors within a specified tolerance given some constraints such as the memory available on the computer used for the simulation. While several error measures are possible, an interpolation error is used in the work presented here. The aim is to make local modifications to an existing mesh, which contains a piecewise polynomial description of a solution field, so that the *existing solution* (as *a posteriori* error measures are required to preempt the solution) is represented optimally for some specified error and computer hardware constraints. The assumption is that the solution will vary sufficiently little within a time interval so that an optimal mesh for some time t_n will also be close to optimal at a time $t_n + \Delta t$. As will be shown later, the *quality* of an element is defined in terms of an error metric that contains directional

information, and so highly anisotropic elements aligned with solution features become possible, and indeed desirable. For example, in boundary layers or fronts the mesh may need to be refined in one direction only. This feature may first appear to be at odds with traditional wisdom in mesh generation that favours isotropic elements. However, in the absence of any mechanism for a mesh to adapt to a solution, isotropic elements may be viewed as being the best compromise.

The remainder of this chapter describes a mesh optimisation method based on that previously described by Pain et al. [3]. In addition, a method is introduced for explicitly preserving the shape of the domain while optimising the mesh. This is necessary for many applications where conservation can be an issue when there are spurious volume changes due to mesh nodes being snapped to, or moved along, parametric models during mesh adaptation. This approach is also used to prevent neighbouring surface elements on slow curving surfaces from being erroneously identified as being coplanar, making it possible for modifications to be applied that may compromise the shape. In addition, a metric tensor gradation control method is introduced, which has the effect of smoothing the metric tensor field, removing sharp changes which undermine the effectiveness of mesh optimisation. A simple adaptive time-stepping method is also described.

2.2 Mesh adaptivity

Fundamental to the method of mesh adaptivity is the formation of a metric which describes in some way the error in the solution. In order to obtain optimal computational efficiency, this metric should not only contain information on the element and node density required to yield the required interpolation errors within the solution, but it should also capture the anisotropic nature of the solution. Elements which are "optimal" in the sense of the metric, may be highly anisotropic. This results in good computational efficiency as one can ensure that such features can be captured with a minimum, yet sufficient number of elements.

In practice, an objective function is formed from the metric so that a single number rep-

resents in some sense the distance of the local mesh from an optimal configuration. The problem of mesh optimisation is then cast as an optimisation problem where the objective function is to be minimised.

2.3 The objective function: Gauge of mesh quality

The objective function used to gauge the quality of the mesh is

$$\mathcal{F} = ||F||_\infty, \tag{2.1}$$

where F is a vector of length equal to the number of elements, and the component F_e of vector F associated with element e is

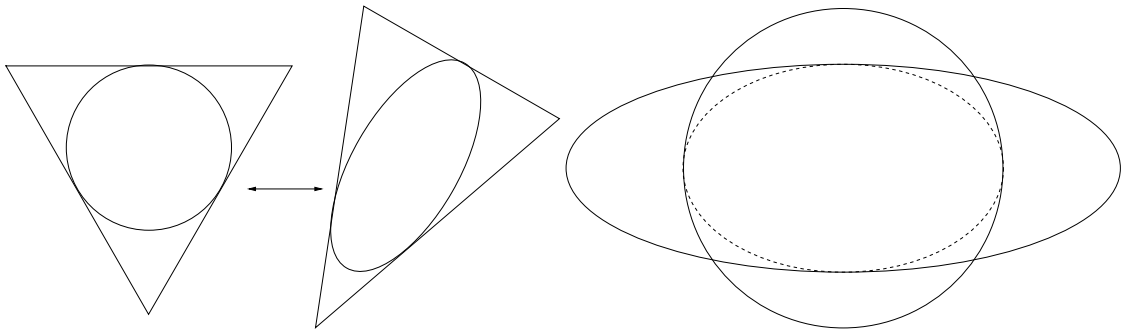
$$F_e = \frac{1}{2} \sum_{l \in \mathcal{L}_e} \delta_l^2 + \mu q_e^2, \tag{2.2}$$

where $\mathcal{L}_e$ is the set of edges of element e and μ is a scalar used to weight the relative importance of the two terms described in detail below ($\mu = 1$ is used here). The first term in (2.2) gauges the size of element e and the second term, involving q_e, its shape. Lower values of F_e indicate *better* elements. Thus, the mesh objective function in (2.1) is actually a measure of the worst element in the mesh.

In (2.2), δ_l is defined for an edge l as

$$\delta_l = r_l - 1,$$

where r_l is the length of the edge l, with respect to the edge averaged metric tensor $\boldsymbol{M}_l(\Omega)$ defined in Ω, see Appendix A.



(a) The mapping (b) Metric superposition

Figure 2.1: (a) mapping between Euclidean metrics; (b) the metric M_1 has been distorted to become a circle and the metric M_2 has been distorted using the same operations. The maximal inner ellipsoid is shown as a dashed line.

The shape of an element, q_e, is defined as,

$$q_e = \frac{\alpha}{\rho_e} - 1,$$

where

$$\alpha = \frac{1}{2\sqrt{6}},$$

ρ_e is the radius of the inscribed sphere of element e in Euclidean space with respect to $M(\Omega)$ (see Figure 2.1(a)). The scalar α has been chosen such that $q_e = 0$ for an ideal tetrahedral element in 3D (aspect ratio of unity in relation to the metric M). The two terms of (2.2) work together in the sense that q_e^2 requires the in-sphere radius of element e to be approximately α while $\sum_{l \in \mathcal{L}_e} \delta_l^2$ when optimised, will push the edge lengths of element e toward unity with respect to M.

The element in-sphere radius, ρ_e, is calculated using the tetrahedral face areas and the element volume in metric space. Numerically, the area of a triangle can be accurately calculated by defining the edge lengths a, b and c such that $a \geq b \geq c$ and computing

$$\tilde{A} = ((a + (b + c))(c - (a - b))(c + (a - b))(a + (b - c)))^{1/2}/4,$$

with the lengths calculated in metric space using (A.1).

The volume of a tetrahedron in Euclidean space can be expressed as one sixth of the

volume of a parallelepiped

$$V_e = det(\boldsymbol{r}_a \boldsymbol{r}_b \boldsymbol{r}_c)/6,$$

where $\boldsymbol{r}_a$, $\boldsymbol{r}_b$ and $\boldsymbol{r}_c$ are any three edge vectors of element e with a common vertex. Note that (6), whose sign is dependant on the order of the three vectors, can also be used to determine the orientation of a tetrahedron. The volume of an element in metric space is then approximated by

$$\tilde{V}_e = (det(\boldsymbol{M}_e))^{1/2} V_e, \tag{2.3}$$

Where $\boldsymbol{M}_e$ is the element averaged metric tensor. The radius, in metric space, of the inscribed sphere of element e can then be calculated using

$$\rho_e = \frac{3\tilde{V}_e}{\tilde{A}_1^e + \tilde{A}_2^e + \tilde{A}_3^e + \tilde{A}_4^e}$$

where $\tilde{A}_1^e$, $\tilde{A}_2^e$, $\tilde{A}_3^e$ and $\tilde{A}_4^e$ are the areas of the four faces of element e.

2.4 Error metric tensor

2.4.1 Definition

In 1D a simple Taylor series analysis shows that the interpolation error ϵ in the representation of a sufficiently smooth function ψ by its piecewise-linear interpolant on a mesh $\{x_i\}$ is given in terms of the function's second derivative. In particular over the mesh cell $[x_i, x_{i+1}]$ define

$$\epsilon = c_0 h_i^2 \left| \frac{\partial^2 \psi}{\partial x^2} \right|, \tag{2.4}$$

where $h_i = x_{i+1} - x_i$ is the local mesh spacing, c_0 is an $O(1)$ constant independent of the mesh ([18], page 348). Equality between the magnitude of the interpolation error and quantity (2.4) can be obtained when the second derivative is evaluated at an appropriate point within the mesh cell, alternatively an upper bound can be obtained by taking a maximum in the right-hand side of (2.4) over the mesh cell.

In higher dimensions, for example on a mesh of tetrahedra in 3D, the analogous result is given by

$$\epsilon = c_0 \boldsymbol{v}^T \boldsymbol{H} \boldsymbol{v} \tag{2.5}$$

for the Hessian $\boldsymbol{H} \equiv \boldsymbol{\nabla}^T \boldsymbol{\nabla} \psi$ evaluated at a point within a tetrahedron, where $\boldsymbol{v}$ is a vector representing a length and direction between any two points within the tetrahedron. The link with the 1D case may be reinforced by noting that the quantity $\boldsymbol{v}^T \boldsymbol{H}(\boldsymbol{x}) \boldsymbol{v}$ is the curvature of the field ψ at point $\boldsymbol{x}$ and in direction $\boldsymbol{v}$. Again a bound on the interpolation error may be obtained by taking the maximum over all points and all directions. The fact that directional information is included in (2.5) is the reason that anisotropic mesh changes may be made.

For a required (specified) global interpolation error, ϵ_g, a metric $\boldsymbol{M}$ is defined such that an element size (length) is unity if it has the desired interpolation error. Thus,

$$\boldsymbol{M} = \frac{1}{\epsilon_g} \boldsymbol{H}^{(abs)}, \tag{2.6}$$

where $\boldsymbol{H}^{(abs)}$ denotes the fact that the eigenvalues of $\boldsymbol{H}$ must be made positive to ensure that $\boldsymbol{M}$ is positive definite. This also reflects the fact that it is the magnitude of the curvature that is of interest rather than the sign of the curvature.

George et al. ([18], page 350) point out that use of a global error norm can in fact alter the error analysis by enhancing the accuracy of regions with high solution magnitude where changes are clearer than those regions of low magnitude. To overcome this, use is made of a *relative error*,

$$\epsilon_r = \frac{\boldsymbol{v}^T \boldsymbol{H} \boldsymbol{v}}{\max(\|\psi\|, \psi_{min})},$$

giving a new metric tensor,

$$\boldsymbol{M}_r = \frac{\boldsymbol{H}^{(abs)}}{\epsilon_r \max(\|\psi\|, \psi_{min})}, \tag{2.7}$$

for some positive ψ_{min}. ψ_{min} is required to be some number greater than zero to avoid division by zero, but it can also be used to suppress curvature values associated with small values of ψ which may be considered spurious. It is important to note that while in (2.6), the interpolation error, ϵ_g, has the same units as the solution field, the relative interpolation error, ϵ_r, is unitless and $\epsilon_r * 100\%$ is the percentage error.

2.4.2 Multi-objective error control

For problems with more than one unknown, each field will have its own metric, reflecting its own required mesh resolution. Each field may also have different error requirements reflected in the metric of that field. The aim is to reduce the problem to the case where just one metric tensor field is necessary. This is done by combining metric tensors together in such way as to protect small edges and directional information. Consider two metric tensors to be merged, $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$, with eigen decomposition $\boldsymbol{M}_n = \boldsymbol{V}_n^T \boldsymbol{\Lambda_n} \boldsymbol{V}_n, n \in \{1, 2\}$ where $\boldsymbol{V}_n$ is the rotational matrix and $\boldsymbol{\Lambda}_n$ is a diagonal matrix containing the eigenvalues of $\boldsymbol{M}_n$. The metric can also be written in terms of its deformation matrices, $\boldsymbol{M} = \boldsymbol{F}^T \boldsymbol{F}$, where $\boldsymbol{F} = \boldsymbol{\Lambda}^{\frac{1}{2}} \boldsymbol{V}$. The first step is to map both metrics to a space where one has eigenvalues of unity, thus:

$$\boldsymbol{I} = \boldsymbol{F}_1^{-T} \boldsymbol{M}_1 \boldsymbol{F}_1^{-1}$$

$$\hat{\boldsymbol{M}}_2 = \boldsymbol{F}_1^{-T} \boldsymbol{M}_2 \boldsymbol{F}_1^{-1}$$

where $\boldsymbol{I}$ is the identity matrix (eigenvalues are all unity). Noting that larger metric tensor eigenvalues correspond to smaller length scales, a new metric tensor, $\hat{\boldsymbol{M}}_2'$, is derived from $\hat{\boldsymbol{M}}_2$ by redefining its eigenvalues, $\hat{\lambda}_i, i \in \{1, 2, 3\}$, as:

$$\hat{\lambda}'_i = \max\{1, \hat{\lambda}_i\}, i \in \{1, 2, 3\}.$$

This new metric is then mapped back to the original space to obtain the final merged metric,

$$M = \boldsymbol{F}_1^T diag(\hat{\lambda}'_i)\boldsymbol{F}_1, i \in \{1, 2, 3\}.$$

The result is illustrated in Figure 2.2. When choosing the metric tensor that will be mapped to the unit sphere it is usual to choose the metric tensor with the smallest aspect ratio (maximum eigenvalue divided by minimum eigenvalue). This protects against degenerate cases such as when merging a 2D disk with a sphere for example. This process is repeated for the error metrics of additional solution fields.

2.4.3 Bounding aspect ratio and edge length

It is useful to modify the metric tensor $\hat{\boldsymbol{M}}$ so as to impose bounds on the maximum and minimum element sizes, as to avoid unrealistic meshing goals. This is possible as one generally has an idea of the requirements of a given problem. For example, the minimum edge length may need to be set to prevent mesh optimisation from trying to resolve eddies below a certain scale, while the maximum should be some fraction of the domain size. Given that the desired edge length in the direction $\boldsymbol{e}_i$ is $h_i = 1/\sqrt{\lambda_i}$, it is clear that the eigenvalues should be replaced by,

$$\tilde{\lambda}_i = \min(\max(|\lambda_i|, 1/h_{max}^2), 1/h_{min}^2) \;\; \forall i \in \{1, 2, 3\}, \tag{2.8}$$

where h_{min} and h_{max} are the minimum and maximum element sizes (more specifically, edge lengths). In addition it may be desirable to control the maximum aspect ratio, a. This is done by decreasing the largest eigenvalue (equivalent to shortening the longest

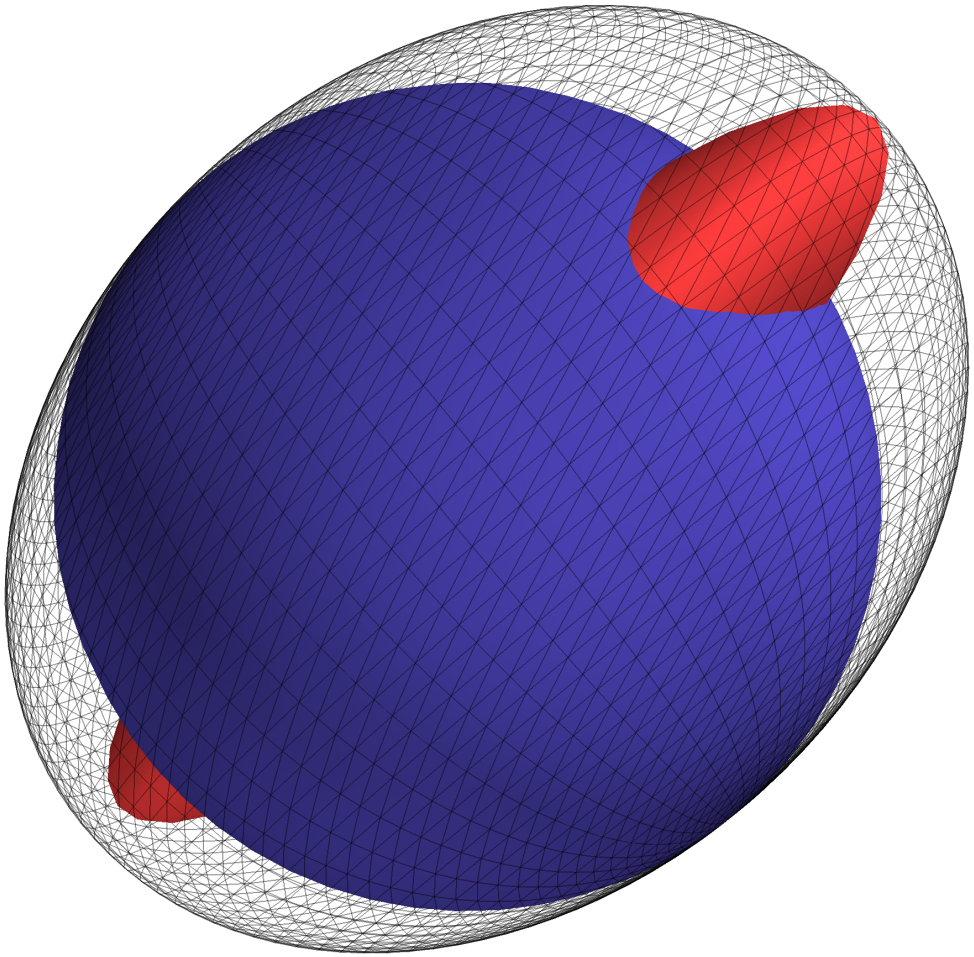


Figure 2.2: The metric tensor $\boldsymbol{M}_1$ is represented as the red ellipsoid and $\boldsymbol{M}_2$ as the blue ellipsoid. The metric tensor $\boldsymbol{M}$, resulting from $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ being combined is represented by the gridded ellipsoid.

desired length scale),

$$\tilde{\lambda}'_i = \min(\tilde{\lambda}_i, \frac{1}{a^2} \max_{j=1}^{3} \tilde{\lambda}_j) \;\; \forall i \in \{1, 2, 3\}. \tag{2.9}$$

However, experience indicates that this is generally not required and has been largely replaced by anisotropic gradation methods (see next section).

Since all the eigenvalues are positive, the new metric tensor is also positive definite. To summarise:

- the maximum aspect ratio for elements is controlled by limiting the smallest eigenvalues with respect to the largest eigenvalue,
- the maximum length scales of elements is limited by imposing a lower bound on the eigenvalue,
- the minimum length scales of elements is limited by imposing an upper bound on the eigenvalue.

2.4.4 Metric tensor gradient control

It is often observed that due to the complexity of both the geometry and physics, and the numerical difficulty of calculating the metric tensor on a piecewise linear mesh, the metric tensor field may include sharp changes in eigenvalues and principal directions. As the mesh optimisation method assumes a well-sampled and differentiable metric space, this can lead to inaccuracies in the mesh optimisation and thus the solution. Recently Li et al. [50] introduced a method for controlling the gradation of a metric tensor field to address this problem. The method, applied in a Gauss-Seidel iterative manner, compares metric tensors connected by an edge in the mesh. The eigenvectors of the respective metric tensors are first paired as to minimise the angle between the two eigenvectors of each pair. A gradation measure on an edge between points P and Q is then defined as,

$$\gamma(\boldsymbol{e}_i^P, \boldsymbol{e}_j^Q) = e^{\frac{|h_i^P(\boldsymbol{e}_i^P) - h_j^Q(\boldsymbol{e}_j^Q)|}{L_{PQ}}} \tag{2.10}$$

where h_i^P and h_j^Q are the desired length scales in the direction of the pair of eigenvectors e_i and e_j associated with the metric tensors at point P and Q respectively, and L_{PQ} is the distance in metric space between P and Q. $\gamma = 1$ signifies a constant field and therefore a regular mesh. A mesh field satisfying $\gamma = 2$ will double in length scale from element to element along the principal direction.

When γ exceeds some tolerance the metrics are modified as to satisfy the gradation tolerance. Importantly, this is performed in such a way as to maintain the accuracy by seeking to satisfy the gradation tolerance by reducing desired edge lengths. The fact that metric tensor eigenvalues are only ever increased also guarantees that the method will not oscillate.

The first step involves rotating the most isotropic metric tensor so that it is more closely aligned with the more anisotropic metric tensor, thereby preserving directional information. Letting R_P and R_Q be the aspect ratio of the tensors, and $R_P \geq R_Q$, the *anisotropy respect factor* is defined as,

$$\alpha = \frac{(R_Q - 1)R_P}{(R_P - 1)R_Q}. \tag{2.11}$$

α is defined in the interval $[0, 1]$ such that $\alpha = 0$ if one of the metrics is isotropic and unity if they have the same aspect ratio. In the case where $R_Q = 1$, the eigenvectors of M_Q can be set to be the case as those of M_P. Otherwise Li et al. suggests setting the new eigenvectors for Q as,

$$e_j'^Q = (1 - \alpha)e_i^p + \alpha e_j^q. \tag{2.12}$$

However this is not exact so it is necessary to rescale $e_j'^Q$ to enforce numerically the unit length of the resulting eigenvectors. Care must be taken when applying this method as eigenvectors may be separated by large angles. A simpler but equally effective approach involves calculating a weighted average metric,

$$\boldsymbol{M}^{pq} = (1 - \alpha)\boldsymbol{M}^p + \alpha \boldsymbol{M}^q. \tag{2.13}$$

The eigenvectors of this metric have the property of being more closely aligned with the more anisotropic metric tensor. The eigenvectors of $\boldsymbol{M}^{pq}$ are then taken to be the new eigenvectors for $\boldsymbol{M}^p$ and $\boldsymbol{M}^q$.

In order to adjust the eigenvalues, let $h_i'^P$ be the reduced size of h_i^P. To make $\gamma = \gamma_0$, where γ_0 is the specified tolerance, (2.10) is rearranged to give,

$$h_i'^P = L_{PQ} ln(\gamma_0) + h_i^Q. \tag{2.14}$$

This process is applied iteratively until all edges have $\gamma < \gamma_0$.

2.4.5 Limiting node numbers

It is important to be able to predict the number of elements that will be in the post-optimised mesh given a metric tensor field, $\boldsymbol{M}(\Omega)$. This is often required for rescaling the metric tensors so that the post-optimised mesh does not exceed hardware constraints (e.g. memory constraints).

The volume of the optimal tetrahedron in metric space is $\gamma = 1/\sqrt{72}$. Assuming that all elements in the mesh are close to optimal after optimisation, and thus have volumes close to γ, then the new number of elements, E_{new}, after optimisation can be estimated using

$$E_{new} = \frac{\sum_{e=1}^{E_{old}} \tilde{V}_e}{\gamma} \tag{2.15}$$

where E_{old} is the number of elements in the original mesh. If $E_{new} \geq \theta E_{max}$ then a new scaled metric is formed, $M_{new}(\Omega) = \beta M(\Omega)$, for some scalar β to be determined. The use of the scalar θ is a consequence of the fact that (2.15) is inexact — however a value of 0.85 has been found to be reliable. When rescaling, the required number of elements is E_{max}, and the following relation can be formed using (2.3):

$$\theta E_{max} = \frac{\sum_e (det(\beta M_e))^{1/2} V_e}{\gamma} = \frac{\beta^{3/2} \sum_e (det(M_e))^{1/2} V_e}{\gamma}, \tag{2.16}$$

giving the result

$$\beta = \left(\frac{\gamma \theta E_{max}}{\sum_e (\det(M_e))^{1/2} V_e} \right)^{2/3}. \tag{2.17}$$

2.5 Mesh Adaptations

For a given error metric and objective function, the tetrahedral mesh is adapted through a combination of:

- node insertion/deletion via edge splitting and collapsing,
- face-to-edge swapping and edge swapping [36],
- Laplacian smoothing (in metric space),
- optimisation based smoothing.

This multi-pronged approach is close to that of Freitag et al. [8] except that the objective function described above is used rather than purely Euclidean measures of element quality.

2.5.1 Edge splitting and collapsing

Edge splitting is a relatively straightforward method for node insertion whereby an edge is split, inserting a node at the mid-point, and bisecting each of the elements surrounding the edge. Edge collapsing on the other hand is a method of node deletion, coarsening the of mesh. This operation collapses the two nodes in an edge to a point in the centre and deletes all the elements that contained that edge. It is noted that one could attempt to find an optimal point along an edge for either splitting or collapsing, however, in practice it is more convenient to allow the r-method (smoothing) sweep to perform this optimisation. In the event that just one of the nodes resides on a surface, the nodes are collapsed to that node. However, care must be taken when both edge nodes reside on a surface. To facilitate discussion, four different nodes are defined based on their position relative to a surface (see Figure 2.3): *internal*, any node that lies inside a surface; *planar*, a surface node where all surface triangles that surround that node are co-planar; *edge*, any surface node surrounded by surface triangles that lie on just two different planes; *corner*, any surface node surrounded by surface triangles that lie on three or more different planes.

Edge type	action
corner, corner	do nothing
corner, edge	collapse to corner
corner, internal	collapse to corner
corner, planar	collapse to corner node
edge, edge	collapse to mid-point only if nodes lie on the same plane, otherwise do nothing
edge, internal	collapse to edge
edge, planar	collapse to edge
internal, internal	collapse to mid-point
internal, planar	collapse to surface node
planar, planar	collapse to mid-point

Table 2.1: Edge collapsing (apply only to nodes that share a common surface triangle)

Given these different definitions, Table 2.1 summarises the possible configurations and solutions. Note that when both nodes are on the surface, it is also necessary that the nodes share a common surface triangle to be considered for collapsing.

2.5.2 Edge swapping and face-to-edge swapping

Face-to-edge swapping can be applied to two tetrahedra that share a common face only if their combined interior is convex. The operation is achieved by inserting an edge between the two nodes that are not shared, removing the shared face and forming three new tetrahedra around the new edge. This operation is illustrated in Figure 2.4. A special case of this exists when two faces connected by an edge are co-planar (the combined tetrahedra form a pyramid and the base of the pyramid is on a plane). In this case the edge can still be swapped but the third, flat, tetrahedron is never formed.

In general, an edge of a mesh can be removed and the surrounding N elements replaced by $2N - 4$ elements. The first obvious example of this is the inverse of the previous face-edge transformation where three tetrahedra surrounding an edge are replaced with two elements. While Freitag et al. [8] established that transformations involving $N > 7$ were rare, the only configurations used in this implementation involve $N = 3, 4$.

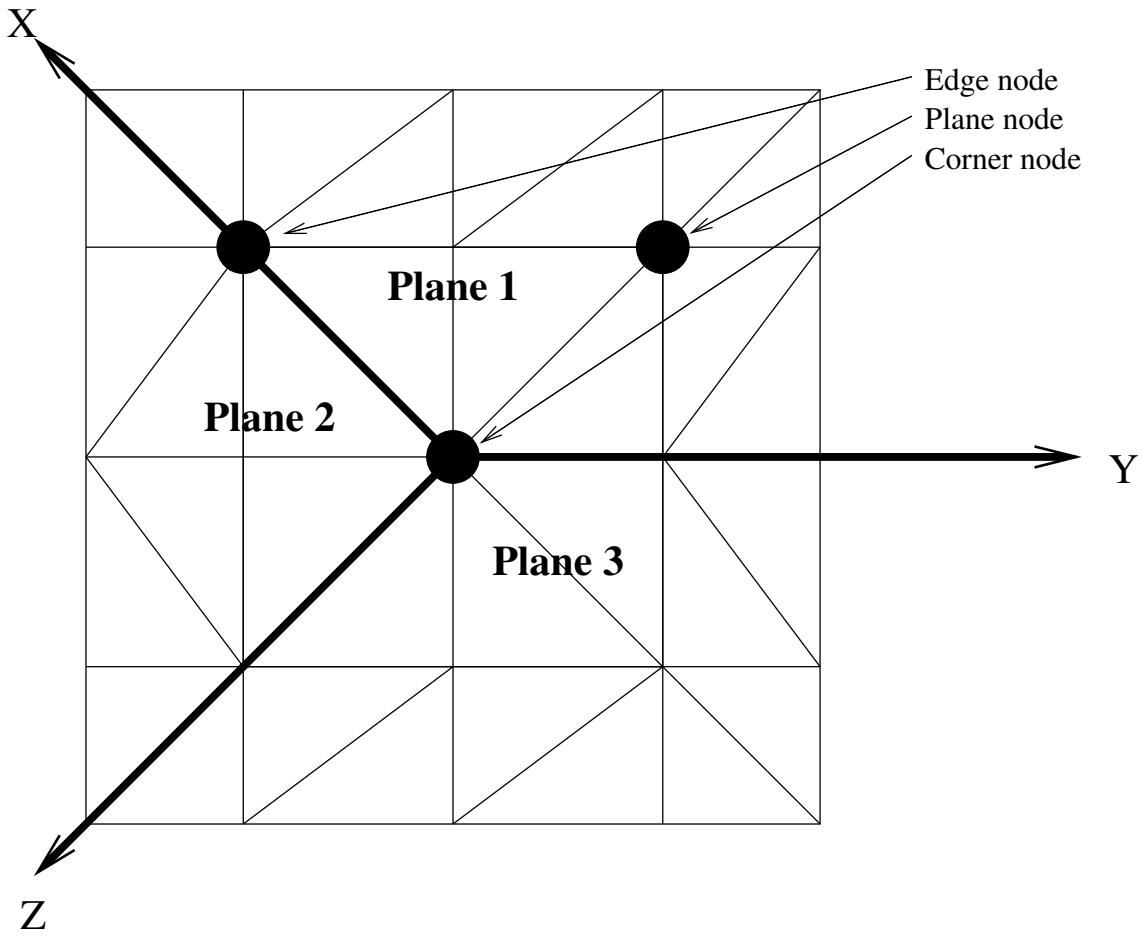


Figure 2.3: Nodes on the surface mesh can be divided into three categories based on local information about connected planes. X, Y and Z are three orthogonal axis and the triangles drawn are on planes drawn between these axis.

2.5.3 Smoothing

The basic smoothing algorithm is a smart Laplacian smoothing in the metric space $\boldsymbol{M}(\Omega)$. That is to say that the mesh is smoothed by minimising the function:

$$E_i = \sum_{l \in \mathcal{L}} r_l^2 \tag{2.18}$$

where $\mathcal{L}$ is the set of edges connected to node i and r_l is given by (A.1). This Laplacian smoothing differs from the traditional approach in that lengths are evaluated in a metric space rather than using Euclidean distances. (2.18) is minimized using a steepest descent algorithm with under relaxation. Before a new node position is accepted checks are made

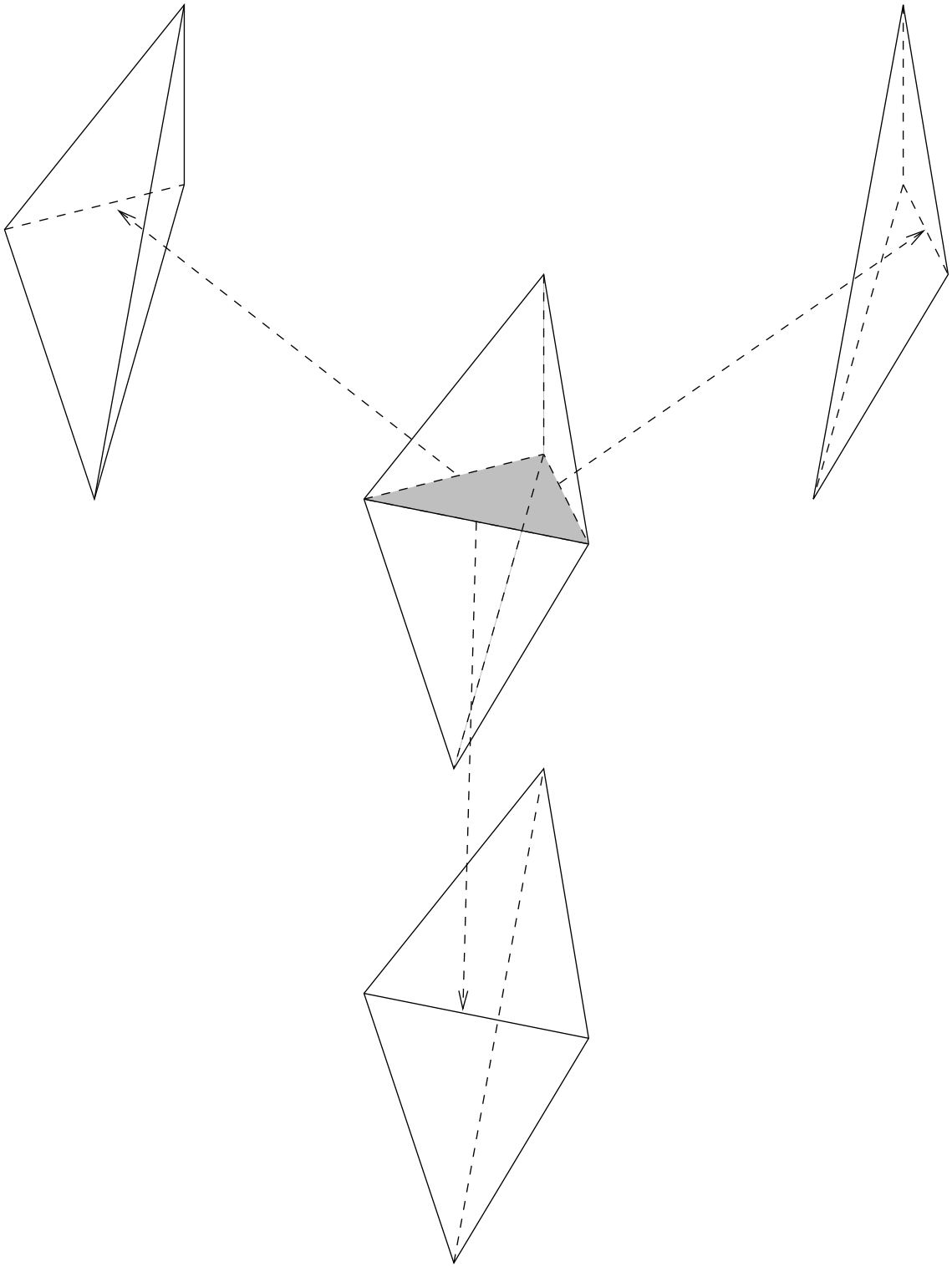


Figure 2.4: Transformation from a pair of elements sharing a single face to three elements surrounding an edge.

to ensure that elements are not inverted by the repositioning. While this algorithm is commonly used and straightforward to implement, Laplacian smoothing has no mechanism to guarantee improvements in element quality.

An optimisation based mesh smoothening method was also proposed by Freitag et al. [8]. This approach optimises the position of each node by seeking to maximise the quality of the worst element connected to the node as measured by some objective function which would be (2.2) for the method presented here. This approach is very attractive as it directly attempts to improve the quality of mesh elements. While it is a more expensive algorithm its contribution to mesh quality justifies its use.

2.6 Surface geometry constraints

A problem with the mesh optimisation method as described is that it is necessary to change the surface mesh as part of the adaptation process. When this is done it is necessary to be able to classify the nodes as done in Section 2.5.1. However, an issue arises with smoothly curving surfaces where adjacent triangles may be co-planar to within floating point accuracy. Thus, triangles can be classified as being co-planar. Over the course of a simulation this can lead to significant "erosion" of the original geometry. A simple solution for many industrial problems is to store the parametric surfaces associated with the simulation. Nodes on the surface are then always mapped back onto that parametric description of the surface whenever an adaptation occurs on the surface. However, this approach can give rise to problems with conservation of mass and other quantities for some sensitive applications such as nuclear criticality simulations. Other applications, ocean modelling for example, have no such parametric description and the discrete representation is usually used.

The approach adopted involves colouring triangles on the original surface representation so that adjacent triangular elements that are co-linear have the same colour. This initial colouring is maintained throughout the simulation to ensure geometry information is not "eroded". This colouring is then used to identify planes as required in Section 2.5.1 or

for moving nodes on a surface. To automate this colouring a region-growing algorithm is employed here. This starts by forming a triangle adjacency list of surface elements (triangles are said to be adjacent if they share a common edge). Two empty lists are then initialised, the *active-list* and the *new list*. An element is selected from the adjacency list, coloured, added to the active list and its normal computed. This will be the *reference-element* for this region. For each element in the active-list, adjacent elements that have not already been coloured are visited and their normals computed. If the normal is equal to the reference-element's normal to within a specified tolerance then the element under consideration is given the same colour and appended to the new-list. When the end of the active list has been reached, the active list is replaced with new-list, new-list is cleared and the sweep through adjacent elements is repeated. Both lists become empty when the region has spread out over a patch of the surface which is co-planar to within the specified tolerance. At this point the colour is incremented and the region growing algorithm is initialised with a new element which has not already been coloured.

The final step in the colouring is to take into account any applied boundary conditions. Different boundary conditions may be applied to different parts of the same continuous co-planar patch. Such patches are recoloured so that only regions that have the same boundary condition applied have the same colour. This avoids "blurring" of the boundary conditions.

2.7 Adaptive time stepping

The use of mesh optimisation methods makes it more difficult to choose a reasonable time step to use for a given simulation. Because of the potentially highly anisotropic nature of elements aligned with the flow, a simple estimate of the required time-step based on the minimum allowed element size and the maximum expected velocity will generally underestimate the maximum allowed time step, thus decreasing the computational efficiency. To overcome this problem, the maximum allowed time-step is continuously recalculated as the simulation progresses. This is done by evaluating the maximum allowed time-step allowed at each quadrature point in the mesh and the minimum of these time steps is

chosen.

The length scale across the element, e, at quadrature point, i, in the direction of the velocity at that point, $\boldsymbol{u}_{e_i}$, is given by:

$$h_{e_i} = \frac{|\boldsymbol{u}_{e_i}|}{|\boldsymbol{J}_e^{-1}\boldsymbol{u}_{e_i}|}, \tag{2.19}$$

where $\boldsymbol{J}$ is the Jacobian matrix:

$$\boldsymbol{J} = \frac{\partial(x,y,z)}{\partial(\xi,\eta,\mu)} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \mu} & \frac{\partial y}{\partial \mu} & \frac{\partial z}{\partial \mu} \end{bmatrix}, \tag{2.20}$$

$$\boldsymbol{J}^{-1} = \frac{\partial(\xi,\eta,\mu)}{\partial(x,y,z)}. \tag{2.21}$$

The maximum time step at a quadrature point i is then calculated from the CFL condition [51]:

$$\Delta t_{e_i} = \alpha \frac{h_{e_i}}{|\boldsymbol{u}_{e_i}|}, \tag{2.22}$$

where α is the maximum Courant Number allowed by the solver being used. The time step can then defined to be:

$$\Delta t^* = \min_{e,i} \Delta t_{e_i}. \tag{2.23}$$

There are two limiting cases that should be considered. The first is the case where the system starts from rest, resulting in $\Delta t_{e_i} = \infty, \forall i, e$. To compensate for this the time step used is the minimum value of two values: Δt_* (as evaluated in (2.23)) and $\beta \Delta t^-$, where Δt^- is the previous time step used and β is the maximum factor it can scaled to ($\beta = 1.1$

is used here). Thus,

$$\Delta t = \min_{e,i} \Delta t^*, \beta \Delta t^-. \tag{2.24}$$

The second limiting case is when the length scales get very small or the velocities get very large. However, this generally arises with numerical instability and thus cannot be corrected without user intervention.

Figure 2.5, from the example detailed in Section 6.2, shows an example of how Δt varied with time in for a simple flow past a cylinder problem.

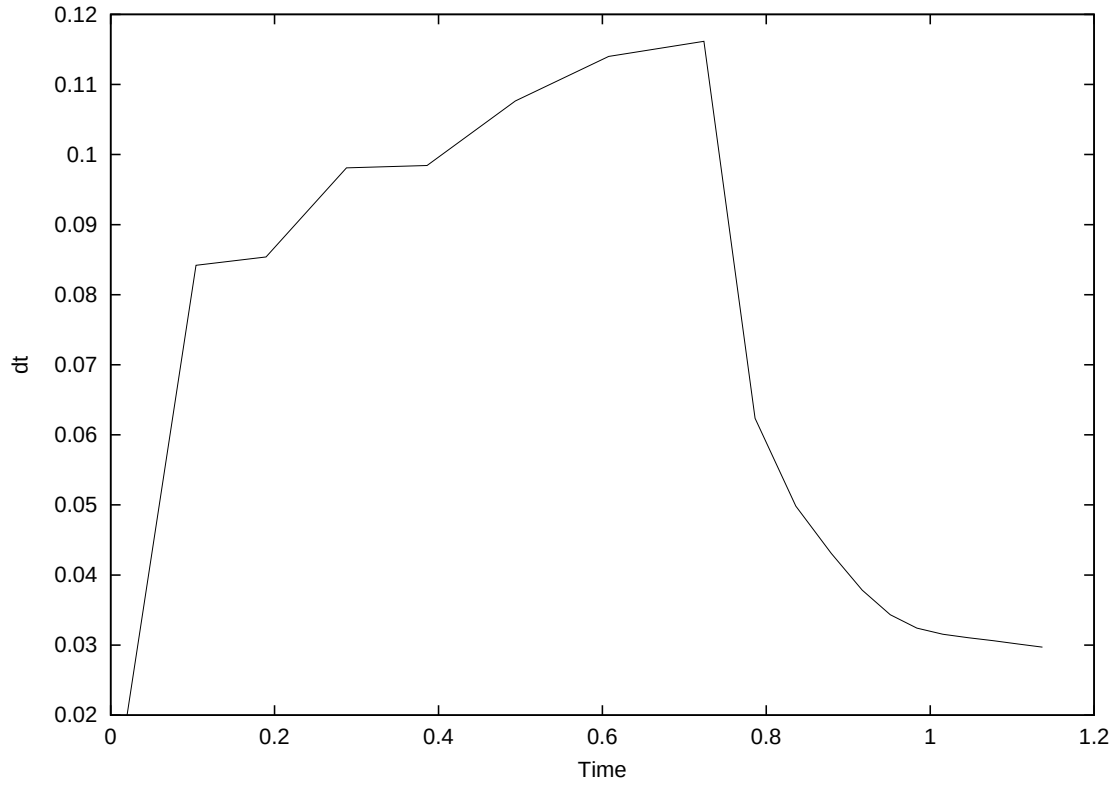


Figure 2.5: Graph shows how the maximum allowed time step varies with respect to time for a simple flow past a cylinder problem when mesh adaptivity is used. The large values for dt at the start of the simulation are mainly due to the fact that the initial mesh resolution is large and the velocities are small.

While this approach was developed for use with implicit methods, it can easily be modified so that spatially-dependent time steps are used, based on the CFL stability condition

for explicit time integration in a fashion akin to Flaherty et al [52]. In such an approach the synchronisation *goal time* (to which the solution is advanced) is usually some small multiple of the smallest time step (2.24). Note however that if this is done the computational load also becomes spatially-dependent (some function of the time stepping) and this needs to be taken into account when load-balancing parallel computation.

2.8 Conclusion

The method presented has been applied by AMCG (e.g. [3, 53, 4, 54, 55]) to a wide range of CFD and radiation transport problems with great success over the last few years. However the field of mesh optimisation and error control is still a very active area of research.

The problem of optimal error control in unstructured meshes can be viewed as a process whereby a measure of error is used to define some space, using a metric tensor, where an isotropic tetrahedral element with edge lengths of unity has a maximum specified interpolation error [9]. As the metric tensor may in general be anisotropic, so too will be the tetrahedral elements when mapped back to the original space. The basic approach is quite flexible and the literature suggests many ways in which the method could be improved further. For example D'Azevedo and Simpson suggest that rather than using the Hessian for the basis of the metric tensor, the square of the Hessian might be used to limit the maximum gradient error rather than the maximum interpolation error [10, 9]. A posteriori error measures, such as sensitivity based measures, are potentially much more powerful for use with mesh optimisation as they into account the physics of the problem, however they are much more challenging to compute (e.g. [5]).

The form of the objective function is also another point of interest, the use of edge lengths and maximisation of in-sphere radius used in this work is only one in a number of possibilities. The work of Freitag [56] and Knupp [44], who also gives a review of different objective functions in use, is particularly interesting in its use of the Jacobian. However the emphasis in the afore mentioned approaches is on generating high quality meshes

without considering the solution and thus meshes tend to be isotropic. One interesting line of investigation would be to re-apply these objective functions, which have been found to be very efficient in Euclidean space, in metric space where isotropic meshes are also sought.

However, for mesh optimisation methods to be of practical use to most applications involved in modelling complex systems, parallel computing is required. While some mesh transformations, such as mesh smoothing, lend themselves to well-known parallel algorithms such as colouring [48, 57] other mesh transformations are much more problematic. It is to this problem that is investigated in the next chapter.

Chapter 3

Parallel mesh adaptivity

Contents

Mesh optimisation is fundamental to computational efficiency. Mesh optimisation aims to achieve a specified *error* throughout the computational domain. This level of control over errors in the solutions allows one to resolve features that might have otherwise been impractical to compute using a static mesh. However, regardless of gains in computational efficiency and processor speed, there is always going to be a need for parallel computing where a solution cannot be obtained within a reasonable timescale on a serial computer and/or where there is a limit on local resources such as memory. Essentially the limitation to serial processing constitutes a computational bottleneck.

The aim here is to parallelise the mesh optimisation method, as described in Chapter 2, so that it may be used with standard DDM methods (e.g. [58]), on distributed memory parallel computers. The parallelisation of the method is vital for it to be applied to large scale complex problems. There are however significant challenges associated with parallel mesh optimisation. These include:

- the adaptation of elements on the interface between sub-domains,
- maintaining the load-balance across sub-domains as the number and distribution of mesh elements and nodes varies.

Generally, an adaptation involves a change in the local mesh configuration. If any mesh entity is to be adapted then all domains that contain information specific to that entity must be updated. The very nature of the mesh optimisation method used precludes the independent prediction on different processors (throughout this text a processor is assumed to operate on a single partition of the domain) of the local connectivity of the final mesh. This poses a problem for the modification of mesh entities at the mesh interface between sub-domains (see Figure 3.2). Local changes to the mesh connectivity are aimed at improving the worst local element and may require modifications to neighbouring entities, which may already be considered optimal. Much of this neighbourhood information will generally be unavailable to elements on sub-domain interfaces. Jones et al. [47] and Freitag et al. [48, 49] describe a sophisticated method, principally based on maximum independent sets, whereby the additional information required by a processor to adapt mesh

entities on the sub-domain interface would be provided by the affected processors, while preventing other processors from performing operations on this region in the other sub-domains. One concern regarding the adoption of this method was that it would require a significant rewrite of the existing serial mesh adaptivity code. More importantly, such a method necessitates a high number of small messages between processors (synchronisations in particular), which incur a relatively high communication cost due to network latency and synchronisations. This is compounded by the fact that the target architectures are distributed memory parallel computers. These facts motivated the development of an alternative solution.

A reasonable estimate of the relative computational load on a processor can be obtained from the number of nodes assigned to the partition on that processor. At the outset of the parallel simulation, graph partitioning is used to define a domain decomposition that balances the number of nodes per processor and thus provides an even distribution of work throughout the processors. However, mesh optimisation operations such as edge-collapsing (node deletion) and edge-splitting (node insertion) disrupts this balance. The induced load-imbalance can range from just a few percent to many orders of magnitude in difference. Because of this, graph repartitioning and data migration (i.e. the redistribution of mesh entities across processors in order to satisfy a new graph partitioning) are key components for parallel mesh optimisation.

While the aim here is not to perform an earnest comparison, it is interesting to note some of the different requirements of parallel mesh adaptivity for optimisation based methods (e.g. Oliker et al. [34]) and hierarchical based methods (e.g. Jimack et al. [32]). In one sense hierarchical based methods are the more straightforward of the two to parallelise efficiently. The error metric dictates the level of the element hierarchy that each element must be refined or coarsened to. This can be determined for sub-domain interface elements in parallel with no communication — though further communication may be required to communicate new edge refinements required to form a conformal mesh when green elements are refined, as described in Chapter 1. This is contrasted with the methods required for maintaining mesh conformity in optimisation based methods (see Chapter 1 and Freitag et al. [49]). However, the advantage of maintaining a complete element hierarchy leads to a complication when load-balancing. The graph partitioning usually

operates on the coarsest level of the graph to avoid the complication of distributing the element hierarchy. The fact that each element of the coarsest mesh represents, potentially, a large work unit which varies between elements can make a good load-balance difficult to achieve. There is also the added cost of migrating these complicated data structures when load-balancing. Naturally, there are many other pros and cons with using the respective methods, but the author is not aware of any study into the relative merits of the different approaches (such a study is well outside the scope of this work) though such a study would be very interesting, particularly for deciding the direction to take for future polyhedral mesh adaptivity development.

The parallel strategy presented here addresses the issues of load-imbalances and the difficulty in adapting elements at the interface between sub-domains simultaneously in a conceptually straightforward, yet efficient, manner. Firstly, the serial mesh optimisation method is applied to each sub-domain independently but disallowing any modification to be made to the sub-domain interfaces. Secondly, load-imbalances will occur because of local refinement and coarsening and so a graph repartitioning is required to re-balance the number of mesh nodes per domain. Later it will be shown how (2.15) can be used to derive a node-wise measure of the relative node distribution after mesh optimisation. This relative measure is used to define node weights on the mesh before graph partitioning, thus load-balance can be obtained before the mesh optimisation operation has completed. In addition to this, weights are applied to the graph edges such that edges associated with poor elements have high weights and are therefore less likely to be split by a partition. This choice of edge weight, when used with diffusive repartitioning methods, has the effect of perturbing partitions away from elements that require optimisation while still load-balancing. Mesh adaptivity is then reapplied to the mesh to address any sub-optimal elements. The procedure is then reapplied if necessary (see Figures 3.1 and 3.5). Because a repartitioning is required to address the load-imbalance caused by mesh adaptivity, the computational cost of this graph partition perturbation and data migration overlaps with the requirements of parallel mesh optimisation. A work flow of the procedure is illustrated in Figure 3.1. Note that this approach to dynamic load-balancing is in contrast to the approach of Oliker et al. where the cost of data migration for a given new graph partitioning is weighed against the computational cost saved if the load-balancing

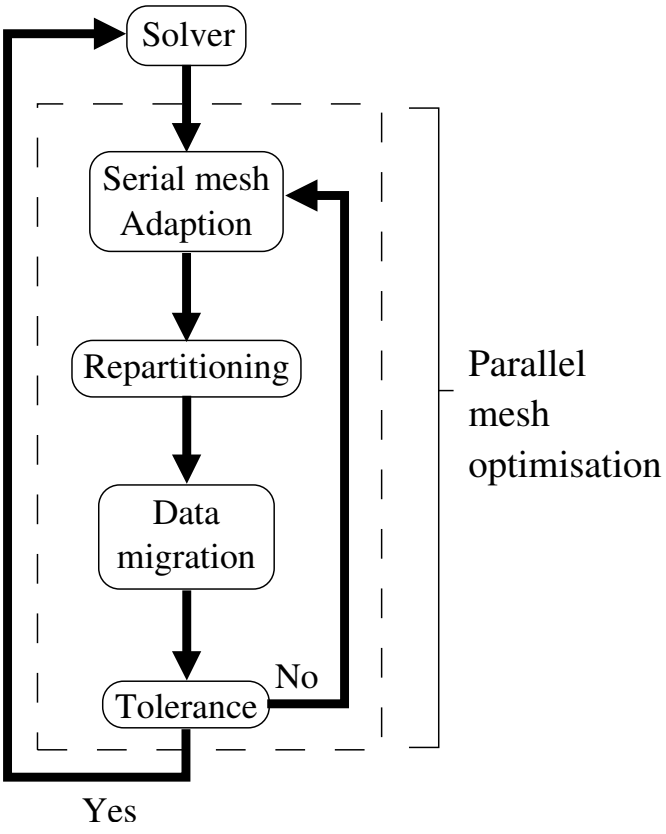


Figure 3.1: Work-flow of parallel mesh optimisation

is performed [34]. However, as will be shown in Chapter 6, the cost of data migration is low when compared to the cost of the solver stage, so the penalty associated with forcing a data migration step will not generally be significant.

The remainder of the chapter is outlined as follows. First the key concepts and operations are introduced. This will provide the basic building blocks required for parallel mesh optimisation and data migration. Next the procedure for graph repartitioning is described. Particular attention is given to the application of suitable node and edge weights. The method developed for data migration is then described in detail. Finally, it is shown how all the components are brought together to perform parallel mesh optimisation.

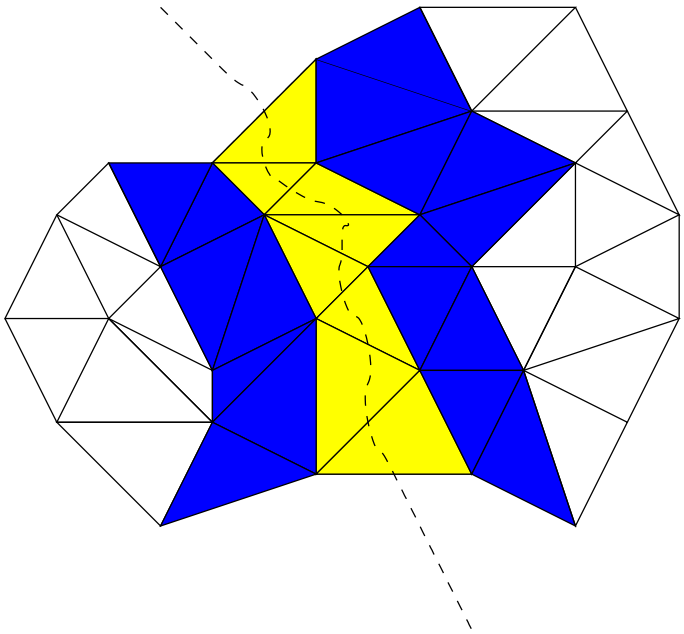


Figure 3.2: Section of the computational mesh: the dashed line indicates the edges cut by the graph partitioning, the yellow region indicates the shared elements (Halo-I elements) and the blue region indicates the extended halo required for the pressure calculation (Halo-II). Thus, the domain on the right includes all clear elements on the right plus all of the coloured region. Similarly for the left domain.

3.1 Building blocks

The purpose of this section is to introduce the basic concepts and algorithms required to develop the data migration and parallel mesh optimisation algorithms in this chapter. All algorithms are presented and implemented in the spirit of SIMD (also known as SPMD), thus should be read as being executed by each processor on their respective sub-domains.

3.1.1 Adjacency lists

Adjacency lists define the connectivity between mesh entities: elements, faces, edges and nodes. When using FEM/FVM the most commonly stored adjacency list is the *element-node adjacency list*, ENList, which for each element in the mesh gives the list of nodes that define that element. As with many examples of adjacency lists, the order of entries in the *element-node* is important as it defines the orientation of an element. The terms

elements and *nodes* used unqualified are taken to refer respectively to a list of all the mesh elements and nodes in a single sub-domain.

A graph derived from a mesh is defined as the *node-node adjacency list*, NNList, where node is synonymous with mesh vertex in this context. This is essentially the list of mesh edges that are attached to each node. Its construction is detailed in Algorithm 3.1.1.

Algorithm 3.1.1: CREATE_NNLIST($ENList$)

$NNList \leftarrow ()$
for $i \leftarrow 1$ **to** $|elements|$
do {
  for $j \leftarrow 1$ **to** $|ENList_i|$
  do {
    $n = ENList_{ij}$
    for $k \leftarrow 1$ **to** $|ENList_i|$
    do {
      if $j \neq k$
      then { $NNList_n \leftarrow ENList_{ik}$
return $(NNList)$

Algorithm 3.1.2: CREATE_NELIST($ENList$)

$NEList \leftarrow ()$
for $i \leftarrow 1$ **to** $|elements|$
do {
  for $j \leftarrow 1$ **to** $|ENList_i|$
  do {
    $n = ENList_{ij}$
    $NEList_n \leftarrow i$
return $(NEList)$

Algorithm 3.1.3: CREATE_EGELIST($NNList, NEList$)

comment: Note that edges do not have orientation

$EgEList \leftarrow ()$
for $i \leftarrow 1$ **to** $|nodes| - 1$
do { **for** $j \leftarrow 1$ **to** $|NNList_i|$
do $\{EgEList_i \leftarrow \{NEList_i \cap NEList_j\}$
return $(EgEList)$

Algorithm 3.1.4: CREATE_EELIST($ENList, NEList$)

comment: { The ordering of elements in EEList$_i$ follows the convention that the j^{th} element in EEList$_i$ is opposite node ENList$_{ij}$. If no such element exists then EEList$_{ij} = 0$

$EEList \leftarrow ()$
for $i \leftarrow 1$ **to** $|elements|$
do { **for** $j \leftarrow 1$ **to** 4
do { $facet \leftarrow ()$
for $k \leftarrow 0$ **to** 2
do $\{facet \leftarrow ENList_i[((j + k) \mod 4) + 1]$
$EEList_{i,j} \leftarrow \bigcap_{i=1,3} NEList_{facet_i}$
return $(EEList)$

The remaining adjacency lists that are important here are the *node-element adjacency list*, NEList (see Algorithm 3.1.2), the *edge-element adjacency list*, EgEList (see Algorithm 3.1.3), and the *element-element adjacency list*, EEList (see Algorithm 3.1.4). Note that there are a number of different ways that EEList can be defined. Here elements are defined

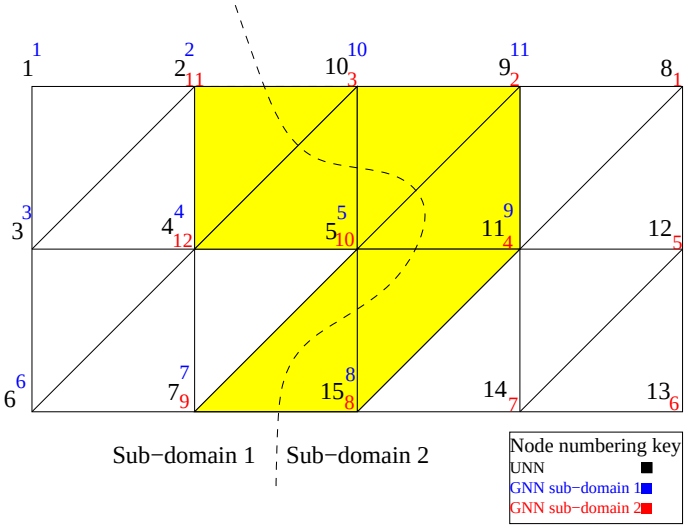


Figure 3.3: Section of the computational mesh: the dashed line indicates the graph partitioning, the yellow region indicates the shared elements. The large font numbering indicates the node numbering over the whole domain. The superscript indicates the node numbering on sub-domain 1 and the subscript indicates the node numbering on sub-domain 2.

to be adjacent only if they share an element face. It is important to note the ordering of the elements in the EEList — the i^{th} element in the list is always opposite the i^{th} node in the element. This is useful when local vicinity searching is employed.

3.1.2 Node numbering

For clarity three different node numberings are defined: the *local node number* (LNN) refers to the local node numbering on an element (for a linear tetrahedron $LNN \in \{1, 2, 3, 4\}$); the *global node number* (GNN) identifies the i^{th} node on a sub-domain; and the *universal node number* (UNN) identifies the i^{th} node in the complete computational domain, (see Figure 3.3). In general, when inter-partition operations are performed on the mesh UNN is used in the adjacency lists (e.g. parallel graph repartitioning).

The pathetic fallacy [1] notwithstanding, it is advantageous to ascribe the notion of *ownership* of mesh entities to processors or domains. Assuming that nodes have been partitioned among processors, a node is said to be *owned* by a processor if it is in the partition

[1] The mistake of ascribing human attributes to objects

assigned to that processor. The owner of an element is defined to be the *minimum node owner*, MNO, within that element where processors are enumerated via their MPI rank number, see Algorithm 3.1.5.

Algorithm 3.1.5: GET_MNO($element$)

$MNO \leftarrow node\_owner(element_1)$
for $j \leftarrow 2$ **to** $|element|$
  do $\Big\{ MNO \leftarrow \min(MNO, node\_owner(element_j))$
return (MNO)

3.1.3 Halo nodes

Halo nodes are mesh nodes that lie on the interface, or overlap region, between sub-domains (e.g. the yellow region in Figure 3.3). There are two classes of halo nodes: those that are owned by the local processors and are also part of a different sub-domain; and those that are owned by a processor other than the local processor and are also part of the local sub-domain. For the purpose of the discussion here, nodes that are owned by processors other than the local processors can be considered to be equivalent to Dirichlet boundary conditions on the sub-domain. As the equations of the system are being solved these values need to be updated regularly. These nodes are denoted $\boldsymbol{R} = (\boldsymbol{R}_1, ..., \boldsymbol{R}_p, ..., \boldsymbol{R}_P)$, where $\boldsymbol{R}_p$ denotes the nodes owned by processor p of P which the local processor receives updates from. Conversely, define $\boldsymbol{S} = (\boldsymbol{S}_1, ..., \boldsymbol{S}_p, ..., \boldsymbol{S}_P)$, where $\boldsymbol{S}_p$ denotes the nodes owned by the local processors that need to be sent to processor p when nodal values are being updated. These lists are constructed in such a way that the n^{th} node in vector $\boldsymbol{R}_p$ on processor q refers to the same node as the n^{th} node in vector $\boldsymbol{S}_q$ on processor p.

3.2 Graph partitioning

Parallel mesh optimisation presents two principal issues. The first is the problem of mesh consistency across processors when optimising elements near or on domain partitions. For example, if a change is made to an element which lies on the interface between domains, the change must be correctly mirrored throughout all relevant sub-domains. The second issue is that of a variable load-balance across processors, due to the fact that mesh adaptivity coarsens and refines the mesh selectively. Such load-imbalances need to be addressed before solvers are applied if the overall method is to be scalable.

The approach adopted here is to use the mesh adaptivity method described in Chapter 2 without modification and to build the required parallel functionality around the serial mesh optimisation method. In this approach the shared elements (yellow elements in Figure 3.2) are first *locked* so that they are in no way modified during the serial mesh optimisation, thus guaranteeing mesh consistency across sub-domains. Each processor then optimises the mesh internal to its associated sub-domain. At this stage all mesh optimisation operations are independent and as a consequence there is no interprocessor communication and the mesh remains fully conforming throughout all the domains. However, due to the fact that the shared elements were locked in place during the adaptive process, there may well exist elements whose quality lies outside the accepted tolerance. In addition to this, the mesh optimisation procedure generally introduces a node imbalance, and therefore a load-imbalance, across sub-domains. Both of these issues may be addressed simultaneously by perturbing the graph-partition in such as way as to move the graph-partition away from regions of the mesh which require modification while at the same time seeking to balance the final number of nodes per partition. This approach is successful because the amount of data that needs to be migrated, and the frequency of migration, are low.

3.2.1 Edge and node weights

A suitable perturbation of the graph partition is achieved through the application of edge weights to the graph. Simply applying high edge weights to regions which were previously locked (because of a graph partition) can encourage unnecessary mesh migration between processors as not all regions which were previously locked are in need of optimisation. The edge weights are derived by first defining the function F_l:

$$F_l = \max_{e|\{l \in e\}} F_e, \tag{3.1}$$

where $F_e \in \Re$, is defined in (2.2). As high values of F_l identify regions of poor mesh quality, high edge weight values are applied to discourage the graph partition cutting these edges. When the graph partitioning calculates the *edge cut* (one of the metrics to be minimised by the graph partition) of a graph partition it sums the edge weights rather than counting the number of edges cut by the partition. However, F_l is a real value, whereas in many implementations of graph partitioning the weights are integer (i.e. multiples of one edge cut) due to the nature of the Kernighan-Lin type refinement algorithm used within ParMetis and many other k-way multilevel graph partitioners, e.g. [59, 60, 61, 62]. For this reason the real edge weights need to be remapped to integer values, and must be suitably distributed so as to ensure that poor elements are effectively impenetrable by the new partitioning (for an element to be cut so too must at least 3 of its edges). The remapping of F_l, to edge weight, W_l used in this work can be written as:

$$W_l = \begin{cases} max(1, \lfloor F_l/F_c + \frac{1}{2} \rfloor) & \\ \infty & \text{if } W_l > W_{90\%} \end{cases} \tag{3.2}$$

where F_c is the objective function tolerance value under which the element is deemed acceptable by the mesh optimisation and $W_{90\%}$ is the 90^{th} edge weight percentile (i.e. 90% of the edge weights fall below this value). This function has the effect of setting

edge weights to be multiples of F_c. As a precaution the top 10% worst edges are given an infinite weighting (in this context the total number of edges in the mesh suffices as a value for infinity) thereby making them unpartitionable. On occasion this may lead to a poor partitioning, usually in the form of sub-domain fragmentation. However, this problem also occurs with the application of diffusive repartitioning techniques and therefore a new repartitioning using an alternative graph repartitioning method (see Section 3.2.2) is periodically required to resolve this problem. Alternative remappings might include mapping the values to a steeper linear function or some more rapidly increasing polynomial, or exponential, however (3.2) appears to work well. It is clear that edges that are surrounded by elements close to optimal are given a weighting of unity and thus unnecessary perturbations to the graph-repartitioning, and therefore data-migration, are minimised. Future improvements to this approach might be to also include information about solution sensitivity or element condition number which may point to regions of the domain that might exhibit slow solver convergence behaviour which would degrade further if partitioned.

While the graph-partitioning is being biased to avoid regions which require further optimisation, it must be kept in mind that a good load-balance is also sought. As mesh optimisation may be incomplete when the graph-partitioning is first performed it is important to be able to estimate the post-adaptation node density. The formulation of the mesh optimisation method makes it straightforward to estimate the local increase or decrease in element density. However, the predicted number of nodes after optimisation is not as readily available. This issue can be resolved by first realising that what is important for load-balancing is the relative weight of nodes with respect to each other. Secondly, a linear relationship between the local number of elements and the number of nodes in the vicinity of a vertex can be assumed. From (2.15) the new number of elements after optimisation in place of a given element can be approximated as

$$\alpha_e = \tilde{V}_e/\gamma.$$

To get a nodal variation of this quantity an alternative expression for the number of new

elements per old element is defined,

$$\alpha'_e = \sum_{j=0}^{n} N_j g_j,$$

where N_j is the linear finite element basis function associated with node j, $\boldsymbol{g} = (g_1, ..., g_j, ...g_n)^T$ is the fractional change of element number at the vertices. g is solved for using a Galerkin projection:

$$\int_\Omega N_i(\alpha'_e - \alpha_e)dV = \int_\Omega N_i(\sum_{j=0}^{n} N_j g_j - \tilde{V}_e/\gamma)dV = 0, \forall i = 1, ..., n,$$

$$\sum_{i=0}^{n} \int_\Omega N_i N_j dV g_j = \int_\Omega N_i \tilde{V}_e/\gamma dV, \forall i = 1, ..., n, \tag{3.3}$$

where Ω is the whole domain. This forms a set of linear equations $\boldsymbol{Mg} = \boldsymbol{b}$, where $M_{ij} = \int_\Omega N_i N_j dV$ and $b_i = \int_\Omega N_i \tilde{V}_e/\gamma dV, \forall i = 1, ..., n$. A satisfactory result can be found by lumping the mass matrix $\boldsymbol{M}$:

$$g_i = \frac{\tilde{b}_i}{\sum_{i=0}^{n} M_{ij}}. \tag{3.4}$$

As was the case with edge weights, the change in the number of elements in the vicinity of vertex v, g_v, must be remapped to be an integer to obtain a vertex weight, w_v. A mapping that works well is:

$$w_v = \lfloor 100 \frac{g_v}{\|\boldsymbol{g}\|_\infty} \rfloor \tag{3.5}$$

This allows a nodal weighting to be applied to the computational mesh so that a graph partitioner can compute a partition balancing the predicted number of nodes rather than the current number of nodes per partition.

3.2.2 K-Way multilevel graph partitioning

The graph partitioning problem involves the division of a graph, $G = (V, E)$, into N partitions, $G_1, ..., G_N$, such that there are approximately the same number of vertices, $|V_p|$, per partition and the number of graph edges, $|E_{cut}|$, cut by the partitioning is minimised. When edge and vertex weights are introduced the problem can be restated in terms of balancing the weight of nodes and minimising the weight of edges cut. Although this problem is in general NP-complete [63], many methods have been developed to find good quality solutions to the problem. Multilevel graph partitioning methods, first applied by [60], have proved to be a popular and effective approach to solving this problem. In the multilevel approach the graph is coarsened to generate a hierarchy of progressively coarser graphs. At the coarsest level the graph can be readily partitioned using for example the *greedy graph growing* algorithm [61]. Subsequently the partitioning is projected back up through the graphs in the hierarchy. Each time the partition is projected back the partitioning is locally refined at the partition interfaces. Exactly how these optimisations are performed is often the principal difference between different multilevel graph partitioning methods. Examples of a few refinement approaches are: artificial neural networks, [64]; Kernighan-Lin type methods, [59, 60, 61, 62]; evolutionary algorithms, [65]. For the work presented here the publicly available code ParMetis [66] is employed which implements a multilevel graph partitioner that can also be used to repartition meshes in parallel (see [67, 68, 69]). It should be noted however that there are important caveats associated with applying standard graph partitioning methods to partitioning domains for DDM with FEM/FVM, see Hendrickson [70]. For example: minimising the edge-cut is not the same as minimising the communication; parallel computation is limited by the speed of the slowest process — thus it is the maximum number of edges in an interface that should be minimised rather than the total edge-count; latency is not considered therefore the number of interfaces is not considered. However these shortcomings are not thought detrimental to FEM applications, [70], and so will not be considered further here.

For the work presented here, the publicly available graph partitioner ParMetis Version 2.0 [66] was used. This uses a fast k-way, multilevel based method which provides a selection of graph-partitioning strategies while allowing the application of weights on both

the edges and vertices of the graph. The three principal parallel repartitioning methods relevant to the discussion here are *scratch*, *scratch-remapping* and *diffusion*. In the first approach a new partitioning is calculated in parallel from scratch using multilevel graph partitioning methods. The resulting graph partition has little dependence on the initial graph partitioning. While this approach gives good quality load-balance and edge-cut it generally leads to significant data migration. *Scratch-remapping* curtails this problem by remapping the partitioning obtained from *scratch* onto the original, maximising overlap between partitions and thus minimising the data migration. The amount of data migration can be further reduced using the *diffusion* method. While this method is fast and results in the least data migration, the method tends to give a higher edge-cut [71, 69]. Indeed it was observed that the partitions can become fragmented after repeated repartitioning using *diffusion*. The strategy is to apply *diffusion* by default to minimise data migration and to apply *scratch-remap* periodically to remove any islands that might occur (every fourth repartitioning in the case of this work).

3.3 Data migration

For parallel computation, where the connectivity of the mesh is subject to change, a basic operation that becomes necessary is the ability to *migrate* subsections of the computational mesh between processors. Data migration (or data remapping) is required to satisfy a graph repartitioning calculated in response to a load-imbalance across processors. The graph repartitioning is expressed as a redistribution of mesh nodes, $\boldsymbol{D} = (d_n), n = 1, ..., N$, where $d_n = p$ specifies the new processor, p, that node n is assigned to. This is a non-trivial operation where an arbitrary node redistribution must be satisfied to maintain consistency across sub-domains and avoiding unnecessary duplication of data. Data migration may be required regularly and may involve a large volume of data, therefore scalability and speed are of utmost importance. In the work presented here, data migration is tightly coupled with the mesh optimisation process. This is in contrast to other implementations found in the literature where the mesh optimisation procedure is fully parallelised and migration is purely used for load-balancing (e.g. [72], [32], [33]). In addition, the data structures associated with the mesh in the optimisation approach are in

general much less complex than those associated with hierarchical based methods due to the fact that there is no mesh hierarchy. This means that the cost of data remapping can potentially be much less than in the hierarchical based methods. However, while the cost associated with data migration might be reduced, it may be that the cost on the solver side is much higher due to the cost of initialising a multi-grid preconditioner if so required for the problem at hand.

Note that mapping everything back to a single processor, as an intermediate step in migration, is not an option because it introduces a serious serial bottle-neck. A single processor may not even have enough memory available to store the complete mesh. Another option is to allow each processor to broadcast its share of the mesh, along with the new decomposition information, to all processors thus allowing each processor to keep what information it requires. This approach is also flawed as there is a high level of redundancy and poor scalability. The aim is to communicate the minimum set of information and to minimise the overall cost of data migration.

The migration routine was designed and coded in an object oriented manner using C++. One of the reasons that C++ was chosen was so that the rich variety of *containers* from the Standard Template Library [73] could be exploited, particularly for sorting data. As will be seen later in this chapter, efficient sorting and indexing are central to the development of the data migration algorithm.

3.3.1 Data-structures

An overview of the class structure used here is shown in Figure 3.3.1. The node and element containers inherit the STL *deque* class [73]. This kind of a container is quick to resize because it does not require continuous memory unlike the STL *vector* class. The *gather/scatter* maps are stored in a sorted set of nodes per processor. By using universal node numbering it is clear that the order of halo nodes in these sets will be consistent throughout the domain, thus maintaining data consistency.

The first operation carried out is to import the mesh and associated data from the relatively

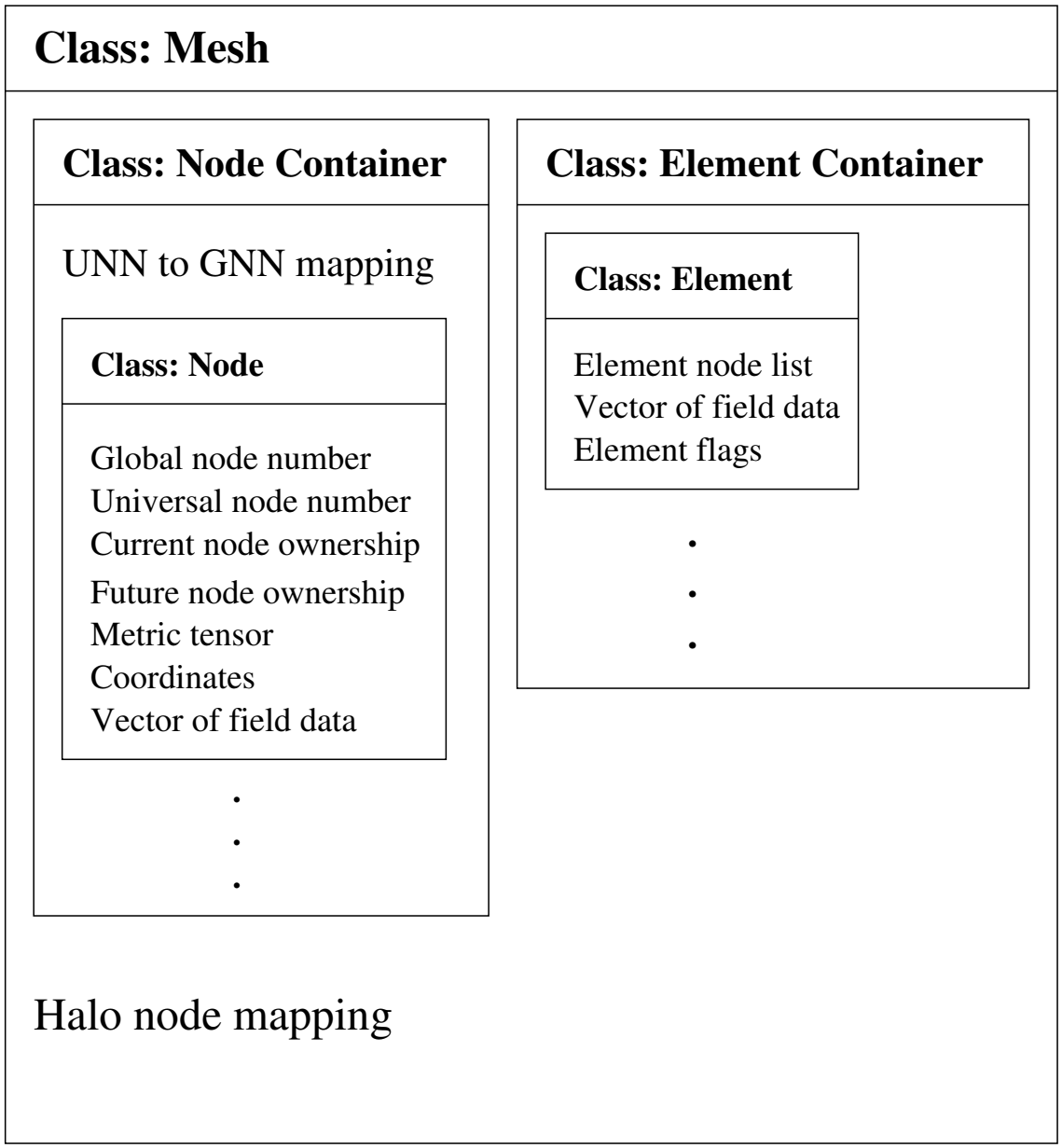


Figure 3.4: An overview of the class structure used in SAM

flat data-structures used by the solver to a more flexible data-structure that allows simple manipulation. Any additional mesh information associated with other discretisations, such as those used in mixed formulations, is discarded at this point. This issue is dealt with in detail in Chapter 4.

3.3.2 Collating existing partitioning with new partitioning

Given the new graph partition, $\boldsymbol{D}$, the first step is to determine the minimum set of data that needs to be communicated in order to satisfy the new mapping. By doing this, inter-processor communication is minimised and no checks need to be done to check for the duplication of mesh objects. We start by considering each element, e, in the domain's mesh. Two logical arrays, $\boldsymbol{A}$ and $\boldsymbol{B}$, of length P are defined as:

$$A_p = \begin{cases} true & \text{if } \exists\ n \in e \text{ where } n \text{ is } \textit{owned} \text{ by } p, \\ false & \text{otherwise} \end{cases}$$

$$B_p = \begin{cases} true & \text{if } \exists\ n \in e \text{ where } D_n \equiv p, \\ false & \text{otherwise} \end{cases}$$

This element is then flagged if it is required on this processor under the new partitioning (i.e. if B_p). Next it is determined which processor is the owner of e. The owner of e can be uniquely defined as the *minimum node owner* of e. If the current processor is the owner of e, then e is listed to be sent to each processor where $(\neg A_p) \wedge B_p$. Now the nodes on the element are checked. If element e has been flagged as a future element then each node in e is flagged as a future node, thus new halo nodes as well as new owned nodes are flagged to be kept. For a node in e to be listed to be sent to a processor p it first must be owned by the current processor and secondly $(\neg A_p) \wedge B_p$ must be true. It should be apparent that each node will be considered multiple times as it is referenced by different elements. To implement this the STL sorted *set* container is used as it allows quick insertion of objects while excluding duplicates.

Define a mesh $\mathcal{M}$ which has been divided into P sub-domains such that $\mathcal{M} = \mathcal{M}_1 \cup \mathcal{M}_2 \cup ... \mathcal{M}_P$. In the context of repartitioning this can be rewritten as $\mathcal{M} = \mathcal{M}_1^* \cup \mathcal{M}_2^* \cup ... \mathcal{M}_P^*$ where $\mathcal{M}_p^*$ denotes the new sub-domain assigned to processor p. In both of these cases the sub-domain includes the nodes and elements in the halo region. The notation $\mathcal{M}_{p'}$ will be used to indicate the sub-domain $\mathcal{M}_p$ excluding the halo elements and those nodes not owned by processor p. In addition $p = l$ denotes the local processor/domain.

Algorithm 3.3.1: CREATE_DATA_TO_PROCESSORS($mesh$)

Initialise ***nodes_to_processors***

Initialise ***elements_to_processors***

for $e \leftarrow 1$ **to** $|elements|$

do
- **comment:** Establish nodes that need to be sent to processors. Check if: n is owned by local processor; n is part of the new sub-domain on p; n is not part of the current sub-domain on p
- **for each** $p \in \{p | e \in \mathcal{M}_p^*\}$
  - **do** **if** $(n \in \mathcal{M}_{l'})$ **and** $(n \in \mathcal{M}_p^*)$ **and** $(n \notin \mathcal{M}_p)$ **then** $nodes\_to\_processors_p \leftarrow n$
- **if** $(get_M NO(e) \equiv p_l)$ **and** $(e \in G_p^*)$ **and** $(e \notin G_p)$ **then** $elements\_to\_processors_p \leftarrow e$

return $(nodes\_to\_processors, elements\_to\_processors)$

At this point we know exactly the minimum amount of information that has to be sent to the other processors in order to satisfy the new graph-partitioning.

3.3.3 Packing and communication

After the information that has to be migrated has been identified, P continuous buffers are allocated. Each buffer is used to pack the sub-mesh that the local processor has to send to each other processor. Once all necessary information is packed, all mesh information on the local processor that is no longer required is deleted. It is important that this is done before initialising communications so as to ensure that the maximum amount of memory is available for allocation when communication begins and the potentially large receive buffers are allocated.

The next stage is to carry out the principal communication of the migration operation. Because of the scale of the communication its efficiency is important. Already because all the data has been packed into a single message, bandwidth penalties associated with small messages are minimised. However, in this case it may be that the length of the message being sent to a processor is zero, but the communication is still important as it tells the receiving processor that it is receiving nothing. The communication is efficiently implemented using non-blocking sends and receives with a preceding blocking probe, Algorithm 3.3.2. First all the non-blocking sends are set up, including those whose message length is zero. Next the number of processors is looped over, each time executing a blocking probe for a communication from *any* processor. On a successful probe the sending processor is identified and the size of the message being received from the processor is read. If the message size if non-zero then a receive buffer is allocated and a non-blocking receive initialised. After all the receives have been set up, all non-blocking communications are collected. As this operation is implemented as a function it is necessary to synchronise processors at the beginning in order to prevent probes collecting communications from the previous call to this function in cases where processors get chronically out of time with one another.

Algorithm 3.3.2: ALLSENDRECV($packed\_data$)

---

for $p \leftarrow 1$ **to** P

  do $\{ MPI\_ISend(data = packed\_data\_p, destination = p)$

for $p \leftarrow 1$ **to** P

$$\textbf{do} \begin{cases} MPI\_Probe(MPI\_ANY\_SOURCE, MPI\_ANY\_TAG) \\ source \leftarrow find\_source() \\ data\_size \leftarrow find\_data\_size() \\ receive\_buffer_{source} \leftarrow allocate\_receive\_buffer(data\_size) \\ MPI\_IRecv(data = receive\_buffer_{source}, source = source) \end{cases}$$

$MPI\_Waitall()$

return (**success**)

3.3.4 Partition reconstruction

At this point, each processor has enough data to construct its new partition complete with its halo nodes. New element and node lists are constructed in-place, while the halo nodes are temporarily stored separately so that they can be added to the end of the node list (a specific requirement of the DDM in the application codes used in this work). Using UNNs the new halos are easily constructed without further communication. For each element shared we loop though each node, n, in that element. If that node is owned by the current processor then every other node in the element owned by processor p, where p is not the current processor, is inserted into $\boldsymbol{S}_p$. On the other hand, if that node is owned by processor p, where p is not the current processor, then it is inserted into $\boldsymbol{R}_p$. Because $\boldsymbol{S}_p$ and $\boldsymbol{R}_p$ are sorted sets, and nodes are referred to by their UNN, then the halo nodes are consistent across all sub-domains.

At this point the mesh has been fully migrated. All that is left to be done is for any send/receive buffers to be freed and the new mesh to be exported back to the solver.

3.4 Bringing it all together

After migrating data with respect to the new graph partitioning the serial mesh adaptivity process can be reapplied if necessary. As only previously locked sub-optimal elements need now be adapted, the computational cost of the subsequent mesh adaptivity operation is relatively low. Whilst in general only one cycle of repartitioning, migration and mesh optimisation is required after the initial serial mesh optimisation, up to three cycles may be required before all sub-optimal elements can be made unrestricted for optimisation (see Figure 3.5). The parallel adaptive cycle is terminated when $\mathcal{F} < F_c$ across all sub-domains, or after four migration-adapt cycles, whichever happens first. It is important to note that because the graph partitioning used node weights based on the predicted number of nodes after mesh adaptivity, the mesh is now also balanced.

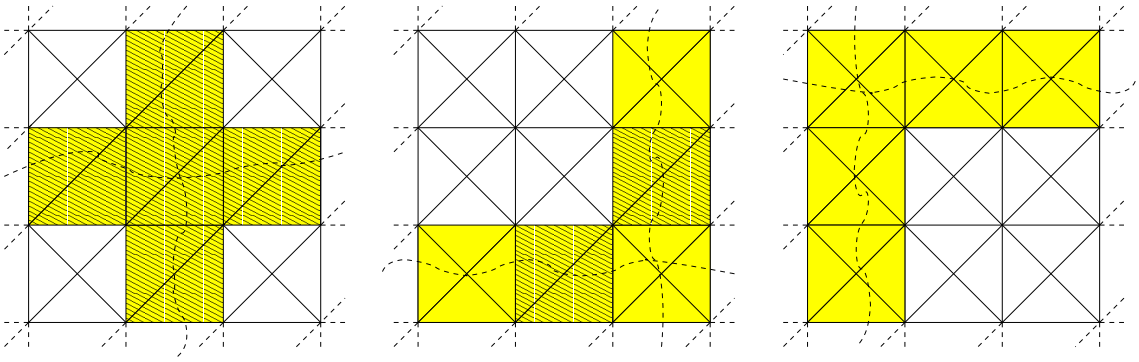


Figure 3.5: In this 2D repartitioning example it is clear that at most 2 successive data remappings are required where the sub-optimal restricted region, yellow hatching, first forms a polyline and then a set of points. In 3D this restricted region would first form a surface and so at most one extra remapping of data may be necessary.

3.5 Conclusions

This work parallelises the serial mesh optimisation method described by Pain et al. [3]. The approach adopted uses the adaptive mesh serial code without modification and to *wrap* around it the required parallel functionality.

Essentially the serial mesh optimisation method is applied simultaneously to each sub-domain, while locking all elements on partition boundaries. From the mesh optimisation procedure, a model for the predicted node density after optimising a mesh for a given metric field is derived. Furthermore an edge-weight is derived from the error metric. It is shown how the node and edge weights are used to control the graph repartitioning so that the computational load is balanced across processors after repartitioning and mesh optimisation, and sub-standard mesh regions are not partitioned, allowing the serial mesh optimisation to adapt the remaining sub-optimal elements.

It is noted that recently Schloegel et al. [69] proposed the *Unified Repartitioning Algorithm* (URA) specifically to address the multi-objective nature of repartitioning when using adaptive mesh methods (implemented in ParMetis 3.1). The URA repartitions a given graph while minimising the objective function,

$$\mathcal{F}_u = |E_{cut}| + \alpha |V_{move}|,$$

where α is the Relative Cost Factor (RCF, also referred to in the ParMetis manual as the ITR), E_{cut} is the edge cut of the partitioning and V_{move} is the total amount of node and element data redistribution required. The RCF describes the relative cost of all the halo specific interprocessor communication with respect to the cost of data redistribution associated with a repartitioning. In practice this is estimated using

$$\alpha = \frac{\text{time to perform all halo communications since last adaptation}}{\text{time to perform all data redistribution in previous adaptation}}.$$

The first time that parallel mesh adaptation is performed, and the denominator of the previous expression is not determined, α is assigned a relatively high value (1000 is suggested) thus favouring *scratch-remap*. This makes sense when one remembers that in general the initial mesh is generated *a priori* to the solution and may in general be far from optimal. Increased levels of mesh adaptation increase the likelihood of severe mesh imbalance and changes to the edge-cut, therefore a clean repartitioning is desirable. While this approach is not used in the current implementation of parallel mesh optimisation, its use will be investigated in the near future and compared with the algorithm presented in Section 3.2.2.

Examples and analysis of the performance of the algorithm are dealt with in Chapter 5. As will be highlighted in the examples, a significant omission in the method as a whole is the lack of a cost model for the serial mesh optimisation algorithm. The lack of such a model means that the parallel method cannot meaningfully attempt to load-balance the work required for serial mesh optimisation. It will be shown that this can be problematic for some problems when there is intense optimisation in just a few localised regions of the mesh.

Chapter 4

Halo communications

Contents

The mesh adaptivity method used in this work is only used with linear tetrahedral elements. The method could in principle be extended to higher order tetrahedral elements by extracting a mesh of linear tetrahedra, applying serial mesh adaptivity to this mesh, re-inserting the higher order nodes and finally interpolating values from the original mesh to the optimised mesh. However, while these steps are straightforward to implement, further work would be required to form a new error metric as the linear interpolation theory presented in Sections 1.2 and 2.4 does not strictly suffice for higher order elements. For this reason the discussion on data remapping in Chapter 3 focused on linear tetrahedral elements when the mesh halo was reconstructed. However, additional tetrahedral based discretisations can still be used. A simple example of these is the *mixed formulation* method used by FLUIDITY [74] when solving for velocities and pressure in the Navier-

Stokes equations. Calculating the FEM mass matrix (each row, corresponding to a node on a mesh) will require not only all information in elements that contain that node, but also information from elements adjacent to these elements. In parallel this means that an extended halo is required, as illustrated in Figure 3.2. Other examples of this are the two geostrophic pressure solvers used by ICOM. The first of these solvers uses quadratic tetrahedral elements. Quadratic tetrahedra have nodes at the tetrahedral vertices and at the centre of the tetrahedral edges (see Figure 4.4). The second method uses discontinuous linear tetrahedral elements, where at each vertex of the mesh there are as many solution nodes as there are elements attached to the vertex. The existence of solution methods that have their own discretisation requirements, that may not even be persistent throughout the solution process, suggests that a flexible approach to constructing and maintaining halos is required. For example, if the particular element discretisation is local to a subroutine and its children, then the cost of reconstructing the halo each time the subroutine is called may be significant when in fact it only needs to be reconstructed after mesh adaptivity. One solution is to pass references to the temporary meshes and associated halo through all the subroutines effected. However, this is not desirable for several reasons. First it involves editing a great many subroutine interfaces each time a new discretisation and associated halo are required by a new solver. Secondly, it increases further the size of subroutine interfaces that are already large and complex. To resolve this issue this chapter describes a module that: stores an arbitrary number of halos in a database; provides a flexible interface for updating halos of solution fields; caches the associated MPI derived data types to minimise startup cost of communications. This is followed by a description of a number of different halos and their construction.

The remainder of this chapter describes the generic halo communication module implemented for the application codes used in this work. This is followed by a description of how to generate an extended halo and quadratic element halo given a underlying linear element mesh.

4.1 Communication module

The principal aims of developing a communication module are to:

- provide a means to *store* and *maintain* an arbitrary number of halos associated with different discretisations;
- dispense with the need to pass all halo information through all the subroutine interfaces;
- provide an interface for performing halo communications given a halo identification number, a data vector and a communication pattern;
- cache all necessary MPI derived data types once created for each communication pattern to minimise the time required to initialise a communication.

This provides a useful level of data encapsulation for which language interoperability is readily achieved.

Generally a halo is stored as a set of vectors containing node numbers. Let us denote the vector of nodes partitioned to processor i and required on processor j as $\boldsymbol{h}_{ij}$. On processor i, $\boldsymbol{h}_{ij}$ defines the node numbers for which field values need to be sent to processor j in a halo communication. Conversely on processor j, $\boldsymbol{h}_{ij}$ defines the node numbers for which field values need to be received from processor i in a halo communication. The node numbering used in $\boldsymbol{h}_{ij}$ is local to the partition, therefore $\boldsymbol{h}_{ij} \neq \boldsymbol{h}_{ji}$. These relationships are illustrated in Figure 4.1. It is possible to define a function, $\mathcal{R}$, that maps a partition wide numbering (GNN) to a unique number for the whole domain (UNN), $\mathcal{R}(\boldsymbol{h}_{ij}) \equiv \mathcal{R}(\boldsymbol{h}_{ji})$. Figure 4.2 illustrates how this information is stored in the halo database. Note that a separate database is used to store halo nodes that are to be received from other processors (nodes that belong to a different partition) and those to be sent to other processors (nodes that belong to the native partition). A dictionary of indexes in the form of *halo tags*, indicated on the far left of Figure 4.2, stores references to each instance of a halo associated with a discretisation. In practice this index is the only information that is required to be passed through subroutine interfaces in the application code. The halo itself is comprised

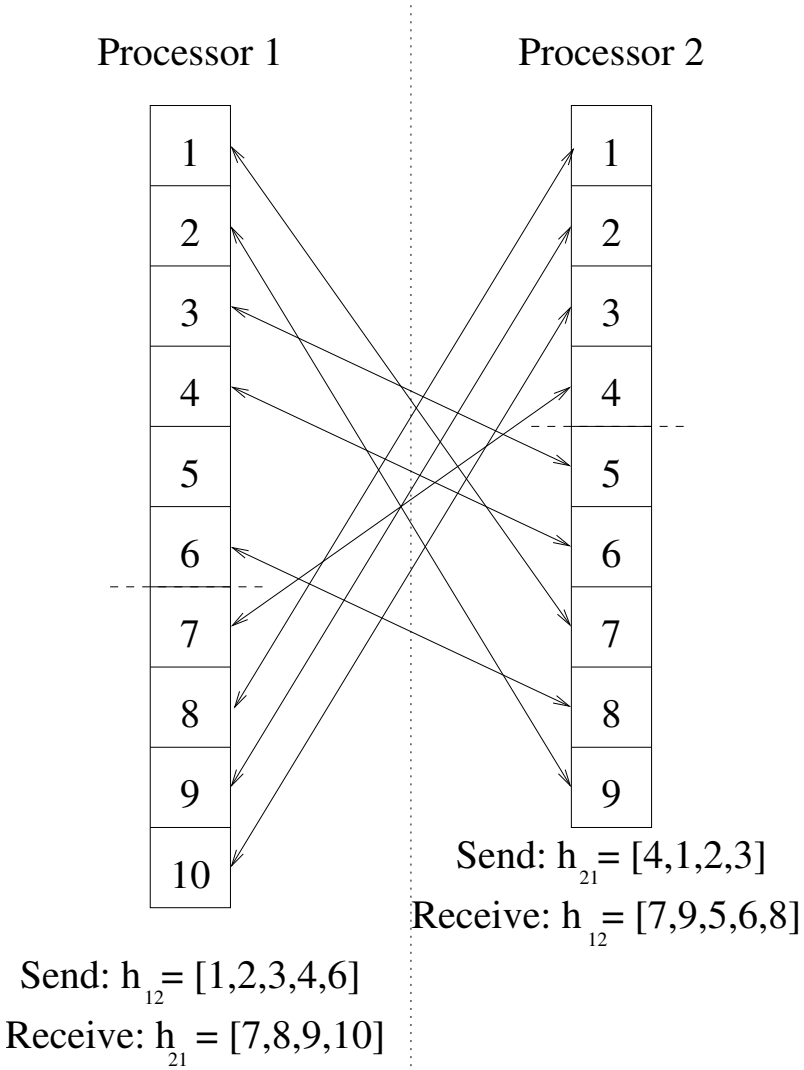


Figure 4.1: Halo vectors between two partitions. In both cases, the node numbers before the dashed line list the nodes assigned to that processor by the graph partitioning.

of an array that contains a reference, for each processor involved in the computation, pointing to an array of node numbers to be received or sent to each processor.

When solution data is stored in continuous memory, as is the case with FLUIDITY, EVENT [75] and many other FEM implementations, most halo communications for a given discretisation can be described in terms of:

- halo node numbers
- the number of field values stored continuously per node

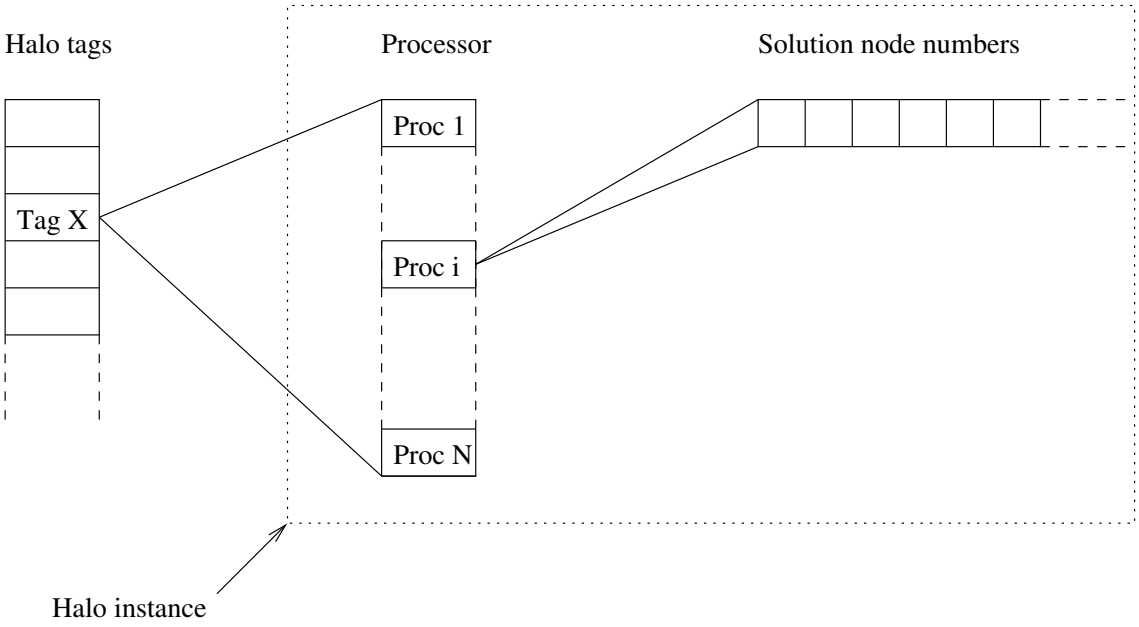


Figure 4.2: Halo database

- the total number of field values to be communicated per node
- the stride in array between nodal field values stored non-continuously

This packing is illustrated in Figure 4.3. The first time that a communication of a characteristic pattern is initialised, MPI derived data types are created and cached for current and future communications. This cache is only cleared after a repartitioning of the graph. Up until recently there has been little advantage in using MPI derived data types as opposed to manually packing the data before sending. In fact recent MPI implementations such as MPICH-1.2.6 use MPI_Pack and MPI_Unpack inside communication routines such as MPI_Send and MPI_Receive. However, the use of MPI derived data types is important as there is scope for MPI implementations to perform optimisations on the packing and the communication which will in general outperform manual packing in the application code, e.g. Byna et al. [76], and ultimately reduce the net communication cost.

The Fortran interface to the halo communication module is as follows:

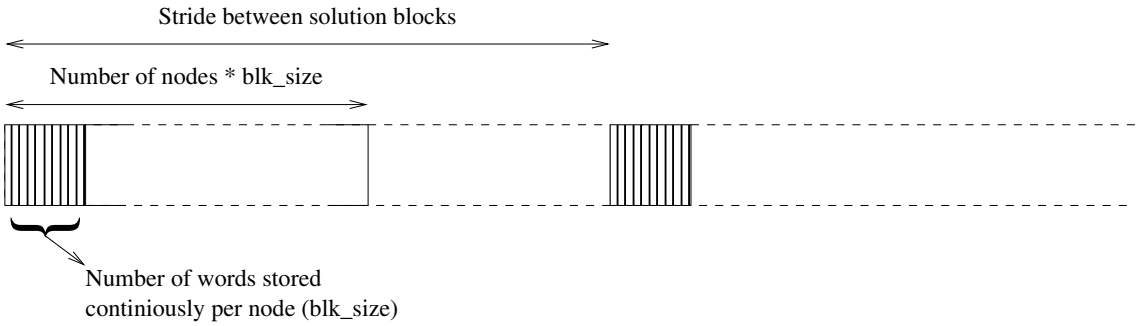


Figure 4.3: Solution variable packing.

```
! Loads a new halo into the  communication module
SUBROUTINE FLCOMMS_REGISTER_HALO(TAG, SEND, SENDPTR, &
                                 RECV, RECVPTR)
INTEGER, INTENT(IN)::TAG, SEND(*), SENDPTR(*),       &
                     RECV(*), RECVPTR(*)
```

TAG halo identifier

SEND, SENDPTR halo nodes to be sent (in compressed storage)

RECV, RECVPTR halo nodes to be sent (in compressed storage)

```
! Update halo ``TAG'' for the solution variables
! stored in V packed in a pattern described by the
! tuple {BLK_SIZE, NFIELDS, STRIDE}
SUBROUTINE FLCOMMS_UPDATE(TAG, V, BLK_SIZE, NFIELDS,  &
                          STRIDE)
INTEGER, INTENT(IN)::TAG
REAL, INTENT(INOUT)::V(*)
INTEGER, INTENT(IN)::BLK_SIZE, NFIELDS, STRIDE
```

V solution

BLK_SIZE number of words stored continuously per node

NFIELDS total number of field variables in v

STRIDE stride between solution fields in v

```
! Delete all halo and cached information. Call before
! new halos are registered after a repartitioning.
SUBROUTINE FLCOMMS_RESET()
```

4.2 Halo types

4.2.1 Extended halo

Previously it was assumed that the halo was defined by elements that were shared across domains. That is to say that they contain nodes owned by different domains. Here these are referred to as Halo-I elements. However, in the case of mixed formulation calculations occurring in some schemes, a larger stencil size can be required to evaluate some of the matrices (e.g. [77]). Looking at Figure 3.2, the yellow elements define the usual mesh overlap, or halo, between two domains. For the mixed formulation, all elements adjacent to yellow elements, are also required on the neighbouring domain. Thus, the total mesh

overlap is the yellow and the blue region combined. The elements in the blue region are referred to as Halo-II elements.

In terms of implementation, the Halo-II is calculated and constructed only at the very end of the parallel mesh optimisation operation. To construct Halo-II we first calculate the Halo-II elements that already reside on the current processor. These are the elements that will define the extended *gather* arrays. It is these elements, along with their associated nodes (which are not already known to the receiving processor) which are communicated.

Define the set $\mathcal{D}_s$ as the set of elements which contain nodes assigned to domain s. Thus the set of Halo-I elements between domains r and s is defined by $H^1_{rs} = \mathcal{D}_r \cap \mathcal{D}_s$, where $r \neq s$.

Next, it is necessary to define an element-element adjacency list on each domain such that $\mathcal{E}_e$ defines the set of elements that share a node with element e. Significant memory is saved by noting that the adjacency relations, $\mathcal{E}_e$, are only required for $e \in H^1 = \bigcap_{r,s} H^1_{rs}$. Halo-II can now be defined as:

$$H^2_{rs} = \Big(\bigcap_{\forall e \in H^1_{rs}} \mathcal{E}_e \Big) - H^1_{rs}. \tag{4.1}$$

The problem with this definition is that it is possible for more than one processor to send the same element to the same destination. In such a scenario an expensive search would be required to remove all duplicates. This occurs for all elements, $e \in H^1 \cap H^2$. The solution to this is to flag all such elements before they are packed to be sent. All the elements and nodes are packed and sent using the *AllSendRecv* operation, Algorithm 3.3.2. When elements and nodes are unpacked, those which have been flagged as being problematic are stored in a sorted set so that multiple copies can be detected and deleted. All other elements and nodes are added into the regular node and element lists. The elements and nodes in the temporary sets are added to lists after all messages have been unpacked.

As the Halo-II nodes are unpacked their UNNs are put into the *extended scattered* sets, S^x_p, where p is the owner of the Halo-II node. The extended *gather* array, H^x, is created

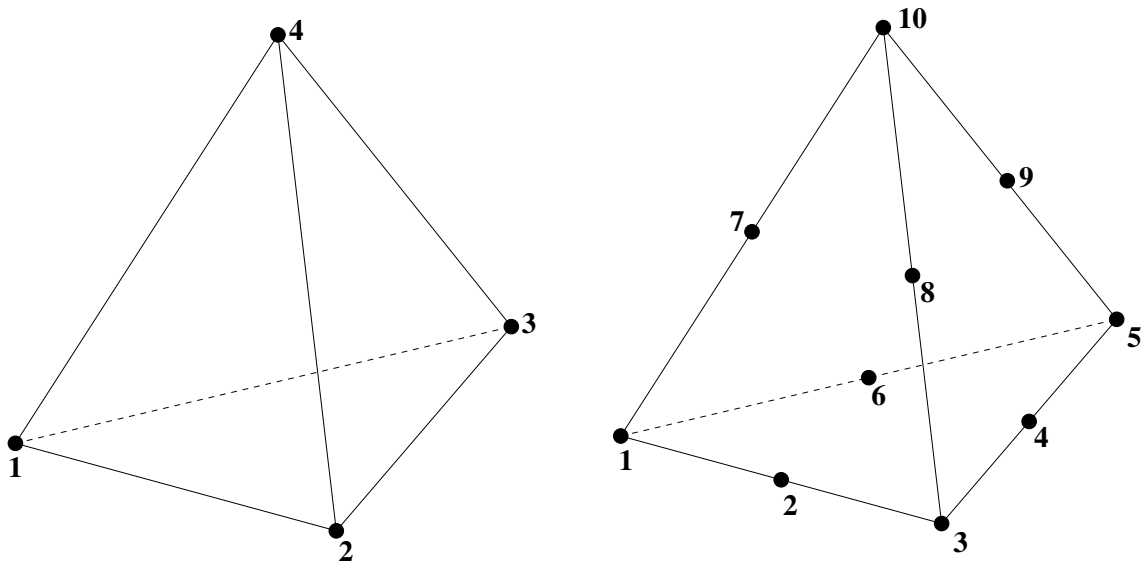


Figure 4.4: Linear to quadratic tetrahedra mapping

by scattering S^x using *AllSendRecv*, Algorithm 3.3.2.

4.2.2 Creating Tetra10 halo from base Tetra4 halo

In some cases a higher order discretisation is required in addition to a basic discretisation to solve for different physical quantities. An example of this is the geostrophic pressure solver used in ICOM. A linear, 4 node tetrahedral element mesh is used to solve the momentum equations and other quantities while a quadratic, 10 node tetrahedral element mesh is required for solving geostrophic pressure.

A straightforward mapping exists from a base Tetra4 mesh to a Tetra10 mesh as is illustrated in Figure 4.4. It is useful to define the mapping that converts the local node numbering of pairs in a Tetra4 element to a local node number in a Tetra10 elements as $\mathcal{T}(i,j)$. Thus, in the example given in Figure 4.4, $\mathcal{T}(2,2) = 3$ and $\mathcal{T}(2,4) = 8$. While it is trivial to generate the Tetra10 mesh from a Tetra4 mesh in parallel, a bit of thought is required for how to construct the associated halo. The aim is to construct a consistent node numbering for the quadratic mesh in parallel, partition the mid-side nodes associated with partitioned edges on the linear mesh and to construct the halo based on this node numbering.

First, a number of sets required for the formation of the Tetra10 halo need to be initialised. Each of these sets are stored in a binary trees to facilitate fast searching:

h_p^t nodes to be sent to or received from domain p;

h^t all nodes to be sent or received by this partition, $\bigcup_p h_p^t$;

h^r all nodes to be received by this partition.

While these sets are being formed, a node ownership look up table is created, $o^n(n)$. The sets of edges shared with each domain, $halo\_edges_p$, and a look up table of the GNN of the mid-side node of each of these edges is created, $mid(e)$. In addition to this a look up table $o^e(e)$ is created which gives the set of domains that require an edge, e. An edge is said to be required by a domain only if an element which contains that edge contains a node that resides on that domain. The algorithm is summarised in 4.2.1.

Algorithm 4.2.1: IDENTIFY_EDGES($halo, Tetra4$)

---

for $i \leftarrow 1$ **to** $|Elements|$
  do if $Tetra4_i \in h^t$
  then
    for each $edge \in Tetra4_i$
      do $insert(o^e(edge), node\_owners(Tetra4_i)$
    for $p \leftarrow 1$ **to** $NProcessors$
      do if $Tetra4_i \in h_p^t$
      then
        for $j \leftarrow 1$ **to** 4
          do for $k \leftarrow 2$ **to** 3
          do
            $insert(halo\_edges_p, Tetra4_i(j), Tetra4_i(k))$
            $mid(e) = Tetra10_i(\mathcal{T}(j,k))$
return (**success**)

The next issue to address is that of partitioning the mid-side nodes. As discussed in Chapter 3, the graph partitioning used in data remapping is based on a graph defined by

the nodes and edges of the linear tetrahedral mesh. It is assumed that once the vertex nodes have been partitioned, neither the load-balance nor the volume of information to be communicated varies significantly if the mid-side nodes in the quadratic mesh are assigned to one or other of the partitions the edge vertices are assigned to. However it must be ensured that the mid-side nodes are partitioned in a consistent manner across all processors. For this the following rule is employed to enforce a unique partition: the partition/owner of a mid-side node is the MNO of the edge vertices. Using this, the sets of edges to be sent/received to/from each processor can be uniquely defined, though unordered, by Algorithm 4.2.2.

Algorithm 4.2.2: PARTITION_EDGES($null$)

---

for $p \leftarrow 1$ **to** $NProcessors$

do for each $edge \in halo\_edges_p$

do
- $edge\_owner = \min o^n(edge_1), o^n(edge_2)$
- **if** $edge\_owner = local\_partition$
  - **then**
    - **for each** $partition \in o^e(edge)$
      - **do**
        - **if** $o^e(edge) \neq edge\_owner$
          - **then** $insert(send\_edges_{o^e(edge)}, edge)$
- **else** $insert(recv\_edges_{edge\_owner}, edge)$

return (**success**)

To ensure a consistent ordering of the halo between sub-domains, so that the n^{th} being sent from processor p_i to processor p_j corresponds to the n^{th} node being received on p_j from processor p_i, a domain wide node numbering is used. First UNNs are generated for the linear tetrahedral mesh giving the LUT $gnn2unn$ which is used to provide a UNN given a GNN on the local processor, Algorithm 4.2.3. Given $gnn2unn$ the reverse LUT, $unn2gnn$, is easily generated.

Algorithm 4.2.3: CREATE_UNN($null$)

$unn\_offset \leftarrow 0$

for $p \leftarrow 1$ **to** $local\_processor\_number$

do $\begin{cases} unn\_offset \leftarrow unn\_offset + |nodes\_in\_partition\_p| \end{cases}$

for $n \leftarrow 1$ **to** $|nodes\_in\_local\_partition|$

do $\begin{cases} gnn2unn[n] = unn\_offset + n \end{cases}$

comment: $\begin{cases} \text{The remaining nodes in the domain are assigned a UNN} \\ \text{by updating the halo values in } gnn2unn \end{cases}$

$halo\_update(gnn2unn)$

return (**success**)

Using gnn2unn the sets $send\_edges$ and $recv\_edges$ can be ordered to provide a consistent ordering across all domains, Algorithm 4.2.4.

Algorithm 4.2.4: COMPARE_EDGES($edge1, edge2$)

---

comment: $\begin{cases} \text{Return:} & 1 \text{ if } edge1 < edge2 \\ & 0 \text{ if } edge1 \equiv edge2 \\ & \text{-1 if } edge1 > edge2 \end{cases}$

$unn1 \leftarrow min(gnn2unn(edge1.node1), gnn2unn(edge1.node2))$

$unn2 \leftarrow min(gnn2unn(edge2.node1), gnn2unn(edge2.node2))$

if $unn1 < unn2$

then $\{$**return** (1)

else if $unn1 > unn2$

then $\{$**return** (-1)

comment: Inconclusive comparison. Check second pair.

$unn1 \leftarrow max(gnn2unn(edge1.node1), gnn2unn(edge1.node2))$

$unn2 \leftarrow max(gnn2unn(edge2.node1), gnn2unn(edge2.node2))$

if $unn1 < unn2$

then $\{$**return** (1)

else if $unn1 > unn2$

then $\{$**return** (-1)

comment: Identical edges

return (0)

At this point a consistent ordering across all domains can be made of the halo vertex nodes and mid-side nodes (mid-side nodes being indexed using the edges) based on UNNs without any further communication.

4.3 Conclusion

The construction of a halo for discontinuous elements was not yet implemented at the time of writing but it is clear that it fits into the framework described in this chapter. Regardless of the discretisation the problem always reduces to two issues: (1) one must define a node numbering that uniquely identifies a single computational node across all processors; (2) the extent of the halo is always determined by the nodes required to assemble the equations for each node assigned to a processor.

The use of MPI derived data types, which themselves only need to be constructed once, means that at worst communications will perform as well as if packing was done explicitly using MPI and sent using MPI_Send/MPI_Recv (or non-blocking equivalents). However the approach affords the MPI implementation to carry out several optimisations related to how messages are copied between buffers. Due to the high frequency of halo communication while using DDM, these savings can be significant.

Chapter 5

Optimal bathymetry approximation [1]

Contents

The accurate simulation of the various processes in a physical system requires an accurate model, domain description, and discretisation in space and time. In the case of ocean

[1] An updated version of this work is available: Gorman et al. [55]

modelling the domain is defined by the bathymetry, coastline, water surface, and any inlet/outlet boundaries. While the surface and in/out boundaries are trivial to model (the boundary conditions are another matter), the inadequate representation of the coastline may lead to problems such as spurious stresses [78]. Additionally, the poor representation of bathymetry can lead to problems in modelling, for example, internal waves or separation of boundary currents (*e.g.* [79, 80, 81, 82]). In addition, poor representation of bathymetry can prohibit through-flows (*e.g.* [83]). In practice there is a trade-off between how close the surface mesh representing the bathymetry conforms to reality, which affects the quality of the simulation, and how appropriate it is for numerical modelling with finite computational resources. Therefore, the mesh should be optimised everywhere with respect to this trade-off, such that the geometry is represented to some specified error with a minimum amount of information or resolution.

In most current numerical models applicable to the oceans the representation of the domain geometry is typically poor with respect to the criteria outlined above. This is a simple consequence of the limitations imposed by the type of point-wise solution representation employed by the majority of models. Many of these models represent the domain with a structured mesh, so that at best curvilinear grid lines yield a mesh of quadrilateral computational cells in the horizontal which may be fitted to the coastal geometry. This horizontal structure is then typically maintained through the vertical levels. The vertical grid generally uses z, sigma(σ), isopycnal/layered coordinates, or a hybrid combination of two or more of these. The most suitable of these coordinates for representing bathymetry is the terrain-following or sigma coordinates. However, large pressure-gradient errors can occur where there are sharp changes in the topography [84]. This problem can be alleviated by smoothing the bathymetry or by increasing the horizontal resolution. These treatments further reduce the user's control of errors in the topographical representation.

Unstructured meshes, by contrast, allow far greater freedom in the representation of complex geometries. In addition, the use of unstructured meshes opens up the realistic possibility of employing adaptive mesh techniques to dynamically alter the local mesh resolution and alignment to best fit the solution characteristics throughout the course of a simulation. However, when mesh adaptivity methods are employed it is necessary to keep the domain fixed to avoid violating volume or mass conservation principles (free-surface and

flooding models also change the domain but aim for conservation). This means disallowing any local operations on the mesh which change the topography, as would occur if the mesh nodes were continually being mapped to a parametric model of the domain for example. Such domain mapping is commonly used in CFD, however conservation becomes more important as the integration time increases. Therefore, since an adapting mesh may seek to refine and coarsen the surface mesh as well as interior mesh, it is important to have as good ('good' being dependent on levels of detail required by the particular simulation in question) a discrete mesh representation of the domain as possible from the beginning of the simulation using a minimum number of nodes/elements. Thus, for this chapter "mesh", unless otherwise stated, will refer to the 3D domain boundary constructed using triangular elements rather than the volume mesh of tetrahedra.

It is important to distinguish between the aims of the work presented here and that of mesh generators commonly used in oceanography. Currently most unstructured-mesh generators, such as *TriGrid* [85] and *Triangle* [86], are based on Delaunay triangulation methods. Delaunay triangulation can also be applied to the surface of a sphere [87]. In addition, many of these methods have facilities to vary mesh node density according to depth and slope, motivated by CFL stability constraints for example. However, many of these features are supplanted when anisotropic mesh adaptivity methods are used in conjunction with the dynamic solution method. In such cases element quality, which may in general have a complex relationship with the equations being solved and the solution [11], is optimised for as the solution evolves. Thus, the original volume mesh becomes largely irrelevant, while the original domain description is paramount.

The approach employed here is to modify and use the anisotropic mesh optimisation algorithm, introduced in Chapter 2. In this application the field which the mesh is being optimised for is the bathymetry (or indeed any height field). The result is an anisotropic mesh which focuses resolution where it is required to optimally represent the bathymetry of the domain. The criterion to judge the quality of the mesh is defined in terms of the Hessian (or curvature) of the bathymetry, which is used to define a general Riemannian geometry, as well as its geometrical merits measured in Euclidean space. An important alternative approach might be to employ a variant of the triangular decimation algorithm [88], where details in a fine mesh are selectively discarded to create less accurate approxi-

mations. However this method does not provide the same control over interpolation error, nor does it facilitate multi-objective mesh optimisation as required when additional scalar fields (*e.g.* material properties, climatology data, or a location dependent importance weighting) also need to be represented accurately on the same mesh.

The remainder of this chapter is set out as follows. In Section 5.1 some motivation for the optimal representation of domain geometry is outlined. In Section 5.2 the bathymetric mesh approximation procedure is described as well as a discussion on coastline recovery within the computational mesh. In Section 5.3 some examples are presented. Finally some concluding remarks are given.

5.1 Motivation

5.1.1 Numerical requirements

When designing an initial mesh for the simulation of a fluid within a particular domain, a decision has to be made as to what accuracy the boundary of that domain should (or needs) to be represented. This accuracy obviously depends both on the underlying physics of the problem in question, as well as the application or equivalently what type of questions the user wishes to answer with the simulation. In oceanographic applications it is unknown exactly to what extent the choice of an accurate boundary representation affects the overall quality of the simulation. It is accepted however that a bad choice can have severely detrimental results. For example, the spurious behaviour present when a piecewise-constant representation of either coastlines or bathymetry is employed. In addition to this, the number of nodes and elements used to represent the domain boundary will directly limit the minimum cost of a simulation, whether it be computed on a fixed or adapting mesh. An appropriate compromise is therefore to seek a representation of the domain geometry to an accuracy sufficient for the user's requirements (perhaps based upon sensitivity studies) using the minimal number of degrees of freedom.

For simulations being carried out with an algorithm able to dynamically adjust its resolu-

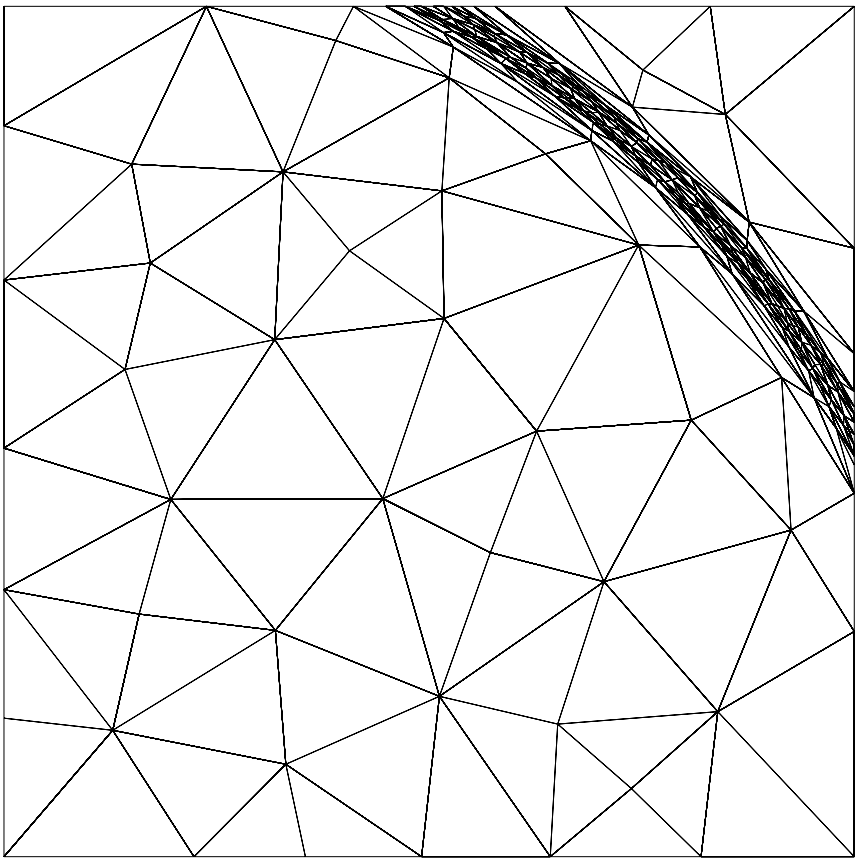


Figure 5.1: An adapted mesh generated for the example in Section 5.1.2. Here an upper limit on the maximum aspect ratio is set at 10:1, and an interpolation error tolerance of 10^{-2} is imposed.

tion throughout the course of an integration in response to the solution, the algorithm may decide that it wishes to refine the boundary representation in some regions, whilst coarsening it in others. Now, in order to ensure that the total volume of the domain is preserved and quality of the domain's geometry is not compromised, coarsening will be (at least in this application) forbidden. This will have an adverse impact on the amount of coarsening that will be allowed in regions close to boundaries. Hence again, it is immensely desirable to optimally represent a given geometry to a prescribed accuracy.

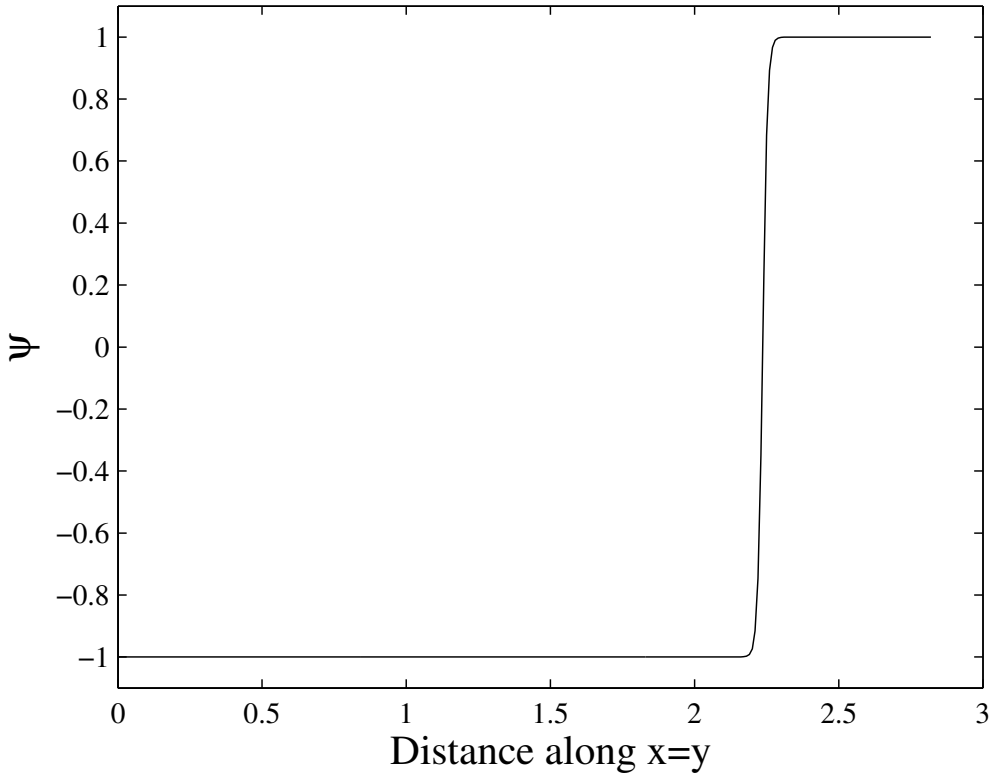


Figure 5.2: The height profile across the diagonal is displayed.

5.1.2 A simple example

In Figure 5.1 a mesh generated for a simple motivational example is presented. Here an initially uniform mesh has been adapted to a given field. The domain is $[-1, 1]^2$ and the field is given by

$$\psi(x, y) = \tanh\left(60\left(\sqrt{(x+1)^2 + (y+1)^2} - \sqrt{5}\right)\right), \tag{5.1}$$

which may be assumed to represent a simple bathymetry field, representing for example a curved shelf region, see Figure 5.1. ψ has been chosen in such a way that it is virtually constant over large parts of the domain, so that large elements may freely be used there whilst maintaining high accuracy. It additionally has a small region of high curvature locally in one direction, whilst the curvature is much smaller in the local orthogonal direction. For example (5.1) the ratio of curvatures normal and tangential to the 'shelf' is approximately $10 : 1$. An optimal anisotropic mesh will therefore be one in which the

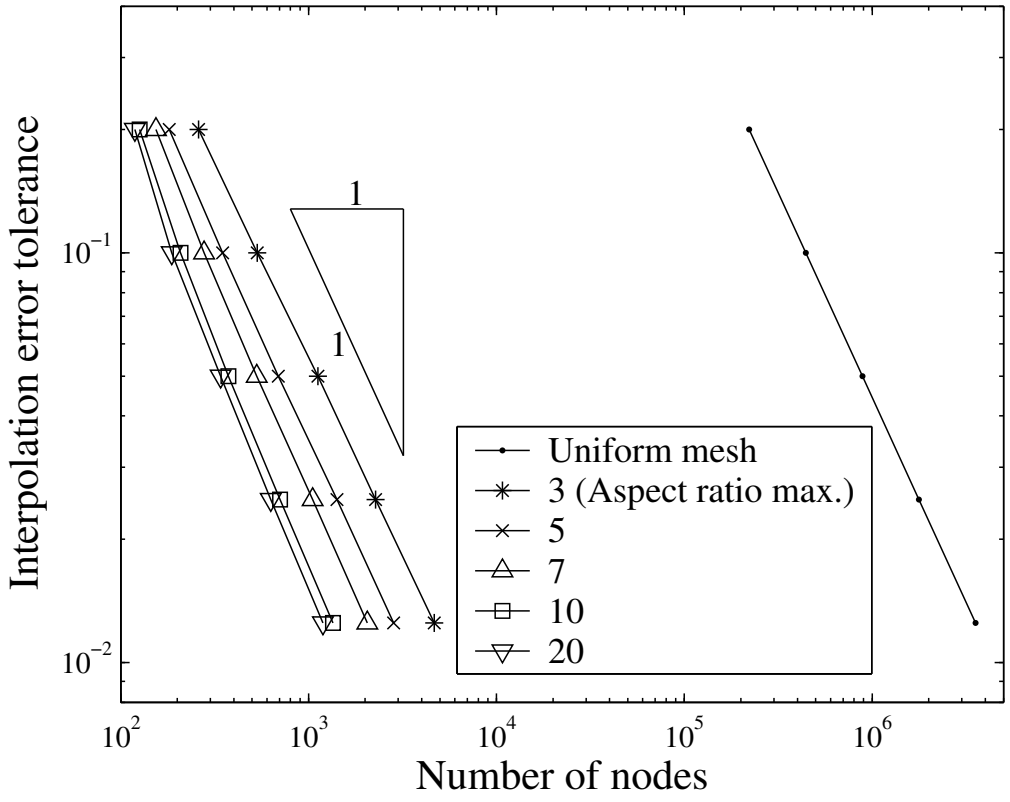


Figure 5.3: Number of nodes required to achieve a given interpolation error tolerance in the representation of field (5.1). Different upper limits on allowable aspect ratios have been imposed. As the limit on element aspect ratio is relaxed the elements have the freedom to become more anisotropic, remaining aligned with the solution. This allows the bathymetry to be efficiently represented using fewer nodes and elements.

elements are 'small' when looking in the direction of the large change in ψ and 'large' when looking in the orthogonal direction, *i.e.* anisotropy is a requirement here for mesh optimality.

To highlight the important effect of anisotropy, in Figure 5.3 a plot is given which shows the number of nodes required to satisfy a given interpolation error tolerance in the discrete representation of (5.1). The meshes were generated from initially uniform structured meshes using the techniques described in Chapter 2. The bathymetry field to be optimised may be considered as a function of (x, y) alone, therefore the three-dimensional (3D) mesh optimisation algorithm is performed in a 2D 'slab' geometry. The number of nodes presented in Figure 5.3 refers only to surface nodes. Each plot in Figure 5.3 corresponds to a different upper limit set on the allowed anisotropy (or aspect ratio) of elements, an

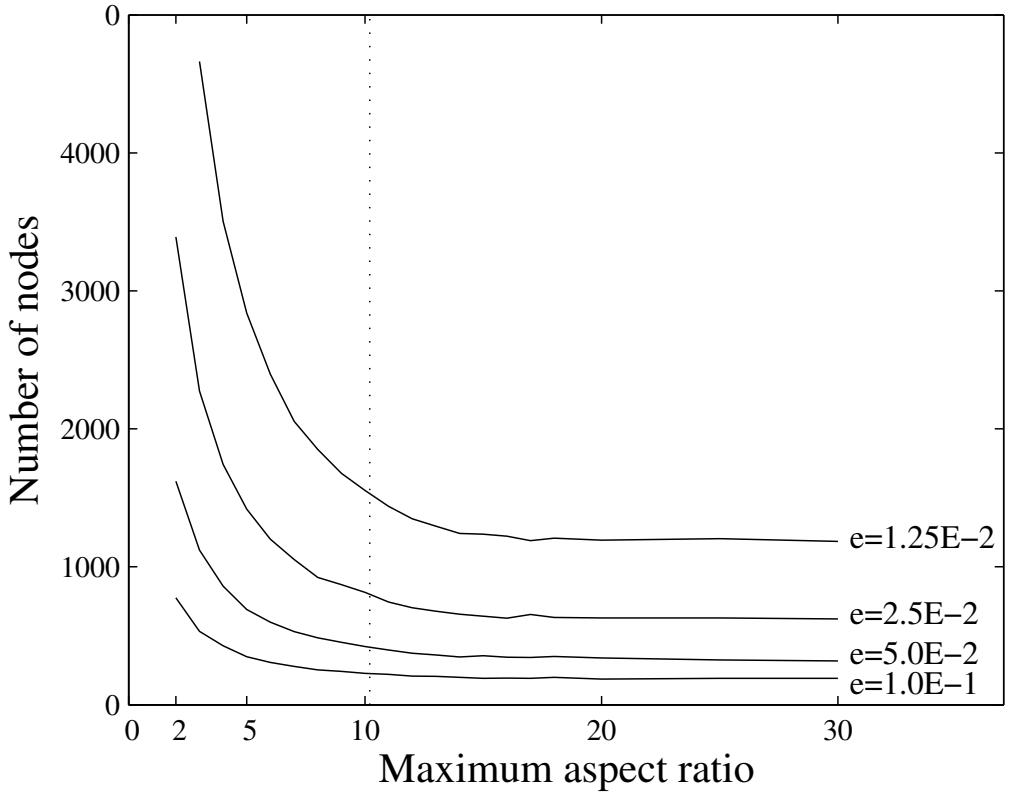


Figure 5.4: The number nodes after optimisation for a given interpolation error and maximum aspect ratio is shown. The actual maximum aspect ratio of the shelf is 10. It is clear from all the profiles that as the maximum interpolation error reaches this value, the required number of nodes quickly tails off.

example mesh with a limit on aspect ratio of $10:1$ and an interpolation error of 10^{-2} is presented in Figure 5.1. As can be seen, allowing high aspect ratios results in a far more efficient representation of the field. As is expected the results scale linearly and allowing an aspect ratio higher than that implied by the local ratio of principal curvatures does not significantly alter the mesh. There is not exact agreement in the meshes at aspect ratios greater than $10:1$ due to the local optimisation based nature of the algorithm, as well as contributing factors such as the presence of boundaries and an imposed maximum edge length here of 0.3. An upper limit on edge lengths was imposed here so that the 'flat' parts of (5.1) are not totally devoid of nodes, no minimum edge length was imposed in these experiments. To complete this comparison a line indicating the theoretical number of nodes required for a uniform structured mesh to respect the interpolation error tolerance is given. This is theoretical purely in the sense that it was obtained from the standard interpolation error estimate with the assumption that the mesh independent constant is unity. This is a fair comparison since these are the same assumptions that the mesh optimisation algorithm are based upon, see Section 2.4.

5.2 Ocean mesh generation

Even when using mesh optimisation techniques with flow solvers, little can be done to improve the representation of the domain geometry while solving. This is because the surface discretisation describing the domain must be calculated prior to any flow computation, and subsequently only nodes and elements that lie on that discretisation may be introduced. Some practitioners of mesh optimisation methods continuously map surface nodes back to a parametric representation of the geometry (*e.g.* [45]). However, due to issues of mass/volume conservation (and conservation of other quantities), it is desirable to keep the same discrete representation of the geometry throughout the simulation. This means that (for isoparametric linear elements) only surface facets that lie on the same plane can be coarsened or refined, see Section 2.6. This provides the motivation to generate a surface mesh of the desired accuracy before computation begins.

5.2.1 Bathymetric source

As optimal shape approximation is of primarily concern here, an initial mesh is assumed to be available. In the case of the examples presented later in this chapter, the initial triangulation was derived from the gridded bathymetry and altimetry data set provided by *GEBCO* [89]. Once the region of interest has been selected, typically specified using a bounding box in terms of longitude and latitude, the gridded data set is triangulated setting the z-component (bathymetry) of the points to zero. The bathymetry is stored as a point-wise scalar quantity. Since the region may contain many points of no interest, only triangles which have at least one node of bathymetry below some specified depth, dz, are generated. Usually $dz = 0$ is used, but this may be set to a value above sea level for coastal modelling with flooding, or below sea level for some ocean circulation models where the shelves and depths are of greater interest than the shallows and true coast. Within this initial triangulation the following non-exclusive types of nodes are readily identifiable:

- *land nodes* – any node of altimetry greater than dz (above sea level),
- *ocean nodes* – any node of bathymetry greater or equal to dz,
- *coastal nodes* – any ocean node connected through an edge to a land node,
- *in/outlet nodes* – any ocean node on the edge of the selected domain that is not connected via an edge to a land node.

Data points that are not included in the triangulation are deleted. In the case where the initial data set is a set of arbitrary points, a Delaunay triangulation is first performed.

5.2.2 Coastline recovery

Careful treatment of the shoreline is required to avoid spurious coastal effects such as those described by [78]. Ideally one would have a suitable discretisation of the coastline and an edge-recovery method would be used to incorporate the boundary into the mesh, *e.g.* using the techniques described by [90]. Furthermore, the procedure should

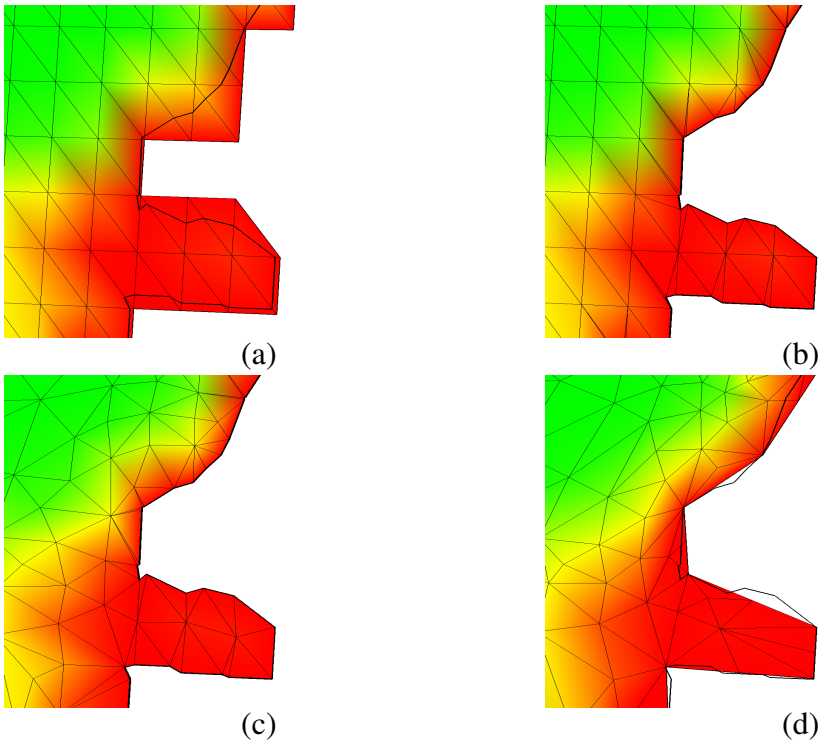


Figure 5.5: The series of images illustrates the steps taken to recover the coastline. The original triangular mesh obtained from gridded data is shown in (a). Triangles are retained only if they have at least one node in the ocean. The black contour shows the position of the coast. The resulting mesh after clipping along the contour is shown in (b). Clipped elements are re-triangulated resulting in many slivers and small triangles. The mesh is then optimised, (c), and finally coastal decimation is performed. The mesh is again optimised to improve element quality on or near the coast resulting in (d).

not require user intervention in the form of editing the coastline by hand as is commonly practised. Not only is this labour intensive but it also compromises the integrity and the reproducibility of simulations carried out on the resulting mesh. In general, coastline data such as the WVS [91] are not suitable for automated boundary-recovery procedures. This is mainly because the shoreline can be self-intersecting, particularly at coarser scales. The authors are not aware of any suitable coastline data set useful for an automated boundary-recovery procedure, and so have developed a simple method with which a satisfactory coastline can be created from the starting data set.

Taking the original triangulated data set as the starting point the coastline contour is first calculated (Figure 5.5 a) as the zero (or dz) level set. Next a clipping algorithm is applied along the coast contour. Clipping involves inserting an edge between each pair of points

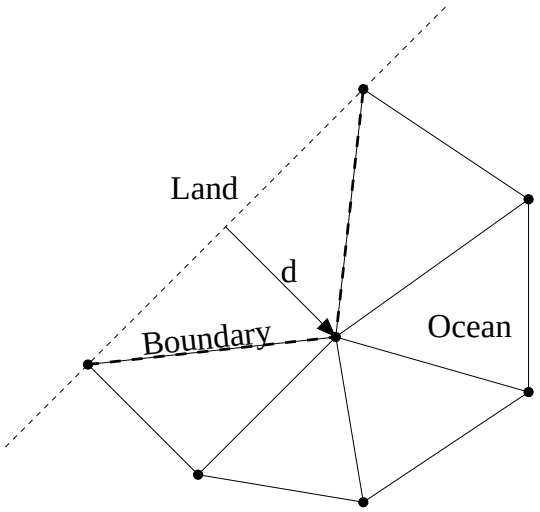


Figure 5.6: The *distance to edge* is the distance between the target vertex (indicated by the arrow) and the new edge formed were that vertex removed.

in each element that the contour enters and leaves, triangulating the resulting polygons, and finally discarding all elements outside the coastline contour (Figure 5.5 b). In general this step results in edges along the coast which are much smaller than the length scale required by the model. This may limit the maximum time step allowed when solving on this mesh. Some data sets might not have data points that are outside the contour and so clipping is not required. However, as the procedure outlined in Chapter 2 tends to lengthen length scales in the domain interior, edges along the coast can be much smaller than those in the internal mesh.

To coarsen the coast an edge decimation algorithm, based on [88], is applied using the *distance to edge* criterion (see Figure 5.6) calculated in Cartesian coordinates (in Euclidean space). Vertices that have a *distance to edge* less than some user specified value are selected for decimation. Decimation is carried out through recursive edge swapping between pairs of elements attached to the vertex in question, until only a single element is attached to that vertex, at which point both the element and vertex are deleted. Sometimes decimation at a vertex fails because an edge swap is impossible without creating a tangled mesh. The frequency of such events can be decreased if the mesh is optimised before the decimation procedure, as the mesh optimisation procedure generally attempts to coarsen the original mesh (Figures 5.5 c and d). Generally, decimation only continues to fail when the internal edge lengths are of a similar length or shorter than the coastal edges and can

be skipped.

The algorithm described also allows the decimation of small islands where all points of the coast have an interpolation error less than that required. This is a desirable result in many applications where very small elements around an island could severely limit the time step. Often it is the case that the inclusion of such small islands may not make an important contribution to the ocean dynamics. In cases where small islands need to be retained the decimation procedure does not remove a vertex if it is part of a three-node island.

It is also important to note that some elements along the coast have no degree of freedom (*i.e.* all element nodes are coastal). These elements may be unsuitable for FE/FV computations as a solution can not develop in these elements. They can either be erased or retained for visualisation purposes.

5.2.3 Optimisation

The bathymetry is treated like a scalar field stored node wise on a 2D mesh. In addition to this, in order to keep track of the nature of each boundary node (coastal or in/outlet node), an integer tag (cast as a floating point number for the purpose of interpolation later) is also stored as a nodal field. Non-boundary nodes are identified using the integer flag zero. As the initial surface mesh is on a plane, the only geometrical constraints on the mesh are those imposed by the 2D boundary, in particular the coast. Now the mesh optimisation method described in Chapter 2 can be applied to the mesh where the field function that the mesh is being optimised for is the bathymetry. For this work a 2D version of the mesh optimisation procedure was not available and so a prismatic 3D mesh, one tetrahedron deep, is formed from the 2D mesh. The height of the slab is set to be the same as the maximum edge length in the horizontal. This limits the effects of capping the maximum aspect ratio, (2.9), when elements are stretched in the vertical. As the resulting mesh is planar, the surface nodes and elements are unconstrained by geometry on the plane. In the 3D case the maximum element size in the third dimension is set to be the extruded height of the domain, (2.8). In addition to this, so not to add information to the mesh, the

minimum element size on the plane is fixed to be half that of the original mesh. Finally, the required bathymetric linear interpolation error, as defined by (2.5), is specified.

After the 2D mesh (or 3D slab mesh) has been optimised, the mesh represents the domain's bathymetry with an equidistributed linear interpolation error. As the mesh is adapted, the metric field values are continuously interpolated onto the new nodes and node positions while the nodal field values (which include the nodal integer flags cast to reals) are interpolated at the end of the adaptive step from the original mesh to the final mesh to minimise diffusion of information. In cases where there is lots of data redundancy it is found to be beneficial to recalculate the error metrics and repeat the optimisation process. The integer tags that describe the nodal types are retrieved by rounding all of the 2D boundary nodal tag values to the nearest integer. The next step is to form a prismatic tetrahedral mesh by extruding the 2D surface mesh. The bathymetric coordinates of the nodes on the top of the mesh are set to sea level while those at the bottom side of the mesh are set to the interpolated bathymetric value. It is important to note that, as in this work, when a 3D mesh is used for the optimisation process, the top or bottom surface of the 3D mesh must be extracted, providing a 2D mesh which is subsequently extruded to follow the bathymetry (the top is chosen here). This is necessary to avoid element vertices falling inside neighbouring elements after extrusion (mesh tangling), which happens following mesh optimisation since the mesh is no longer prismatic. This is particularly problematic in regions of the mesh where the bathymetric interpolation error in the raw data set is higher than the one specified for mesh optimisation.

The extension of the method to the globe is relatively straightforward. The key point is that in order to apply the above algorithm the mesh must be projected to a 2D plane. In the case of the globe this can be done using the stereographic projection,

$$\begin{aligned} x &= 2\tan\left(\frac{\pi}{4}-\frac{\alpha}{2}\right)\sin(\beta), \\ y &= -2\tan\left(\frac{\pi}{4}-\frac{\alpha}{2}\right)\cos(\beta), \end{aligned}$$

where α is latitude, and β is longitude. For simplicity a spherical globe is assumed. The

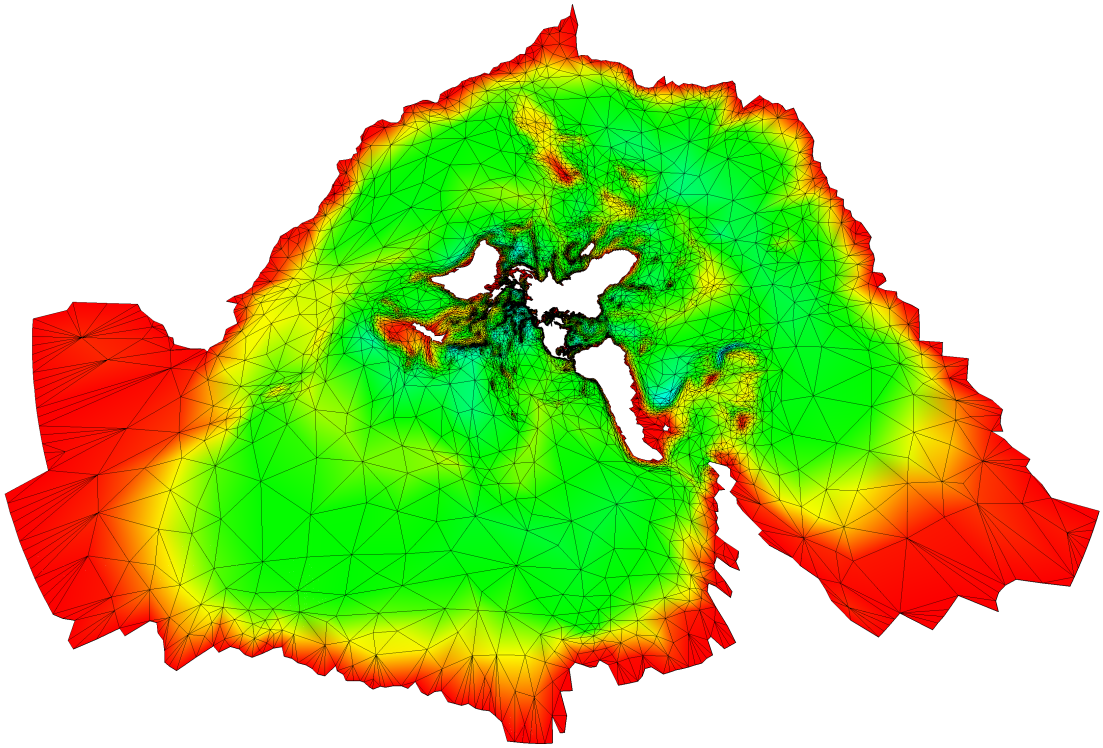


Figure 5.7: A stereographic projection of the globe and resulting optimised mesh, where 90° south is mapped to infinity. This is a conformal mapping which means that angles are preserved between this space and the surface of a sphere (Compare with Figure 5.10).

South Pole is at a point at infinity, which does not cause a problem in this application since it is on land and therefore never meshed. Figure 5.7 shows an example of the projection. An important property of the Stereographic projection is that it is a conformal mapping (*i.e.* it preserves angles), [92]. The fact that distances are distorted in this space is irrelevant because distances are measured in a metric space using (A.1), thus no distortion is evident when the mesh is mapped back to a sphere. Hence element quality, as measured with respect to the bathymetry field, is preserved.

5.3 Examples

Some examples of unstructured variable resolution anisotropic meshes constructed with realistic topography shall now be presented. Along with each example a brief description of the benefits of high quality meshes conforming to coastlines and topography is given.

5.3.1 The North Atlantic Ocean

It is often noted that ocean general circulation models find obtaining the correct structure and separation point for the Gulf Stream at Cape Hatteras problematic. In many models the Gulf Stream tends to continue further north along the coast after Cape Hatteras before separating. This has serious consequences with regards the model's ability to simulate accurately circulation in the North Atlantic. Many mechanisms/agents that may influence the generic properties of boundary current separation have been suggested, for a review see [93]. Two important agents are coastline orientation and the details of the local topography [94, 95, 79, 81]. In addition, it has been shown that increasing the model resolution alleviates this deficiency, see for example [96, 97, 98]. Consequently, the ability to resolve the coastline and local topography to a resolution higher than that employed in the interior, or less dynamic regions, of the simulated domain represents a considerable advantage in circulation models. Optimal bathymetric representation has the advantage of allowing investigations into the sensitivity of ocean circulation patterns to variations in topography, results of which will undoubtedly lead to guides for the level

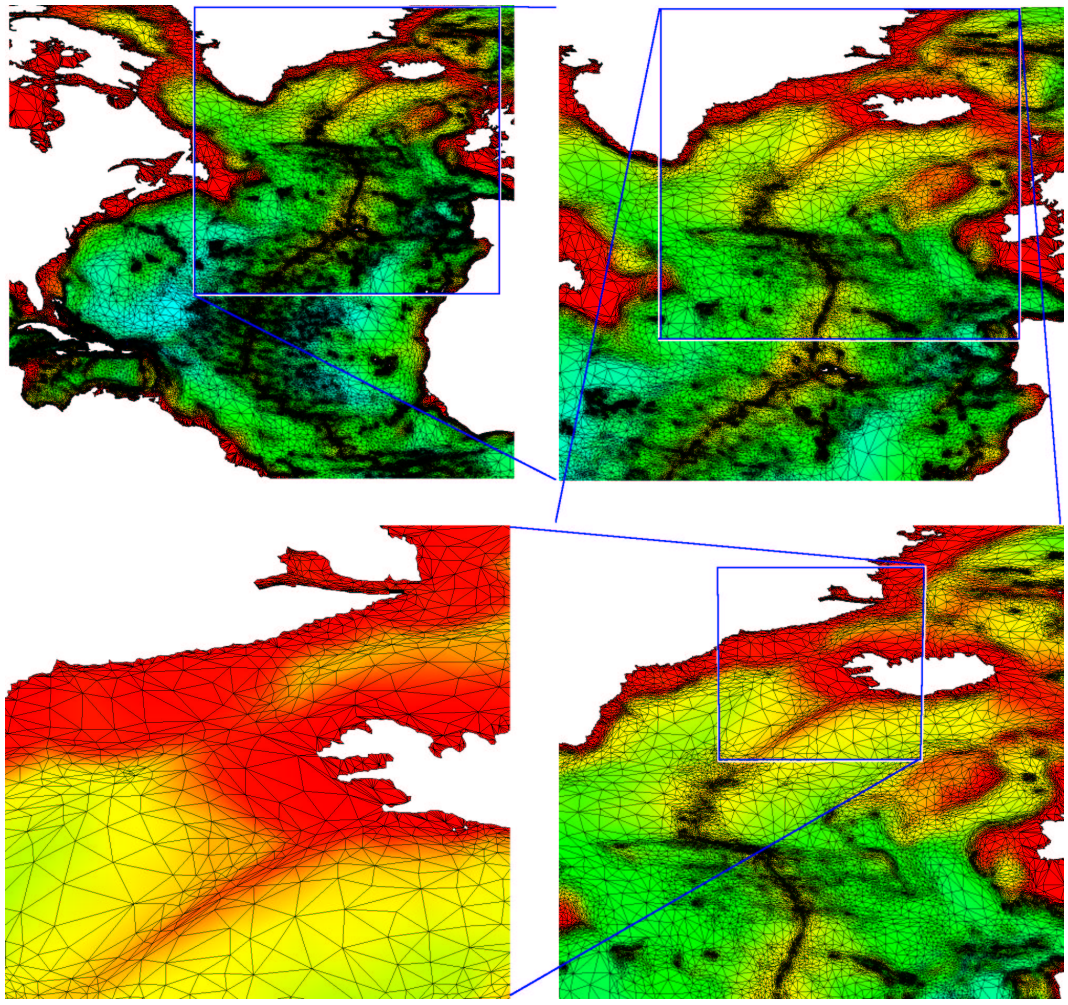


Figure 5.8: Mesh of the North Atlantic with approximately 100k nodes and a maximum resolution of $\frac{1}{10}^{\circ}$, this being the resolution the raw data was sampled at and thus the minimum element size.

and type of resolution that is needed in operational simulations. Figure 5.8 shows a mesh generated for the North Atlantic, included are sections of the Greenland and Labrador Seas. Another example of the importance of bathymetry detail on flow dynamics is the topographic steering of dense overflow water through the Denmark Strait. This has consequences for the dynamics of the thermohaline circulation and subsequent implications to climate change. The ability to accurately model these processes relies on the ability to resolve the topographic features to a sufficient accuracy. So that, for example, important canyons or straits do not fall within the sub-grid scale, and the slope of the shelf break may be correctly modelled on the particular mesh employed. For examples of studies on these issues see [99, 100]. Related to these points are the investigations done through the DOME group [101].

5.3.2 The Mediterranean Sea

The Mediterranean Sea possesses an extremely complex coastline strongly influencing circulation, as well as bathymetry which has fairly large regions of uniform depth, but with many small islands and a few regions where the bathymetry sinks to great depth. Along with the general circulation patterns and eddies, the dynamics within the Mediterranean Sea includes many varied processes which are strongly influenced by the details of the coastlines and topography. For example, overflows of dense water into the North Atlantic [102], convection in the Adriatic, as well as deep convection in the Gulf of Lions. The fact that open boundaries are small and can very often be ignored, along with the quality of observations, makes this an excellent region for testing numerical simulations. An example of a mesh for this region is given in Figure 5.9.

5.3.3 The globe

As a final example a mesh constructed for the spherical Earth is given. The bathymetry data is mapped via the stereographic projection to a slab upon which the mesh is constructed, see Figure 5.7. The mesh is then mapped back via the inverse of the stereo-

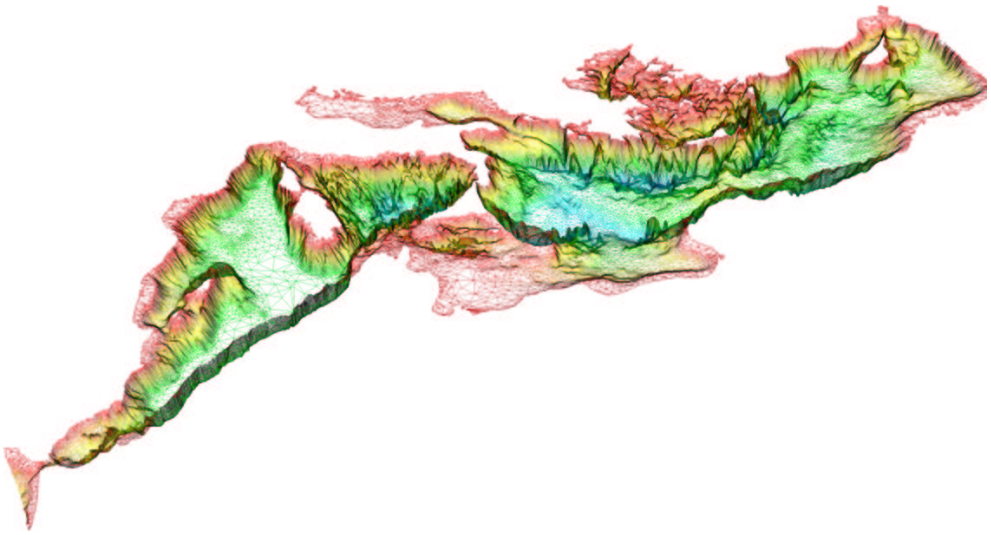


Figure 5.9: An isometric view of the bottom surface mesh of the Mediterranean Sea. The vertical has been exaggerated here by a factor 50 for visualisation.

graphic projection to the sphere, as shown in Figure 5.10.

5.4 Conclusion

Mesh adaptivity methods can be used to generate an optimal representation of a given bathymetry to a specified interpolation error. This provides a mesh upon which an unstructured mesh control volume or finite element based calculation can be performed. It is also possible to optimise the mesh with respect to an interpolation that varies in space, thus defining the interpolation error as a function of horizontal position. It is also possible to include an interpolation error as a function of depth. An example of an application of this could be to set the interpolation error at a point to be some percentage of the depth so as to give more importance to shallower regions. Ultimately a sensitivity analysis will be required to analyse the relationship between errors in the representation of the bathymetry and those in the solution. While a coastline recovery algorithm is effective, many improvements are possible. In particular the literature contains many improvements to the original decimation algorithm described by [88] which could be incorporated into the algorithm used here. Further developments of the technology presented here are also possible, for example, climatology data used to initialise the model could also be opti-

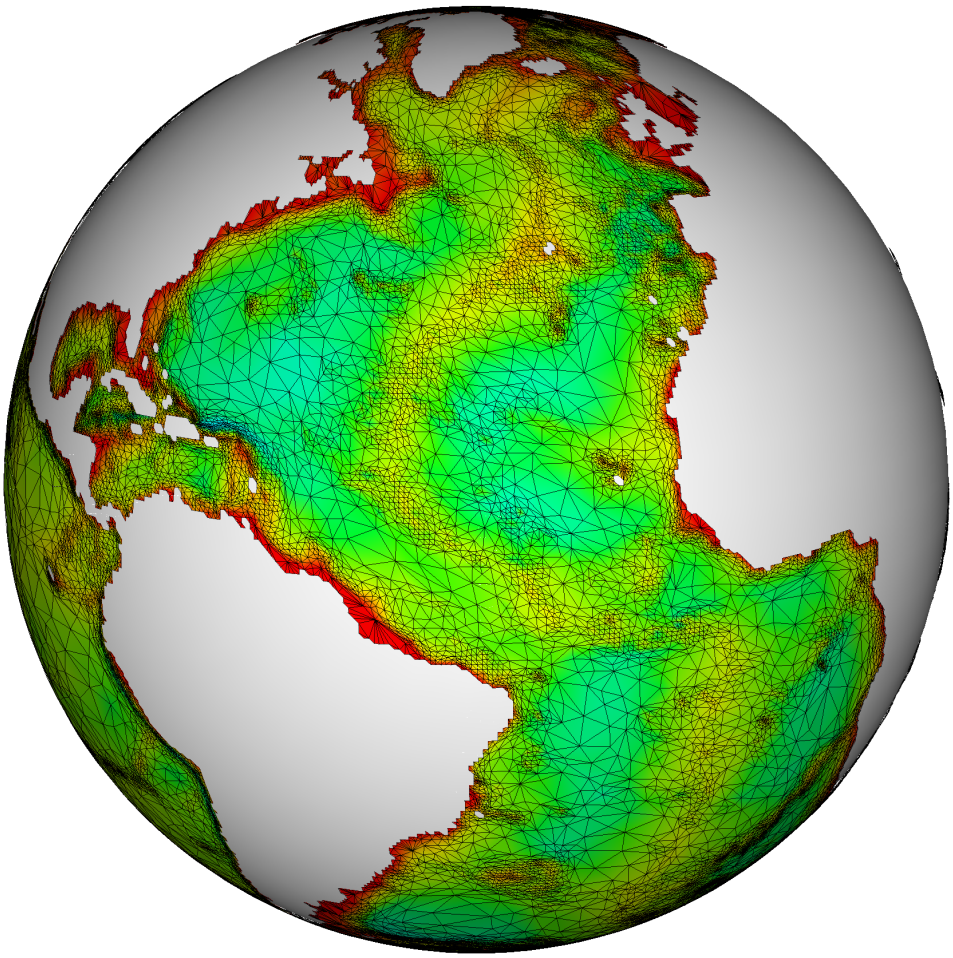


Figure 5.10: A mesh of the globe (compare with Figure 5.7).

mised for in the mesh generation process. This would also present a good approach for setting the initial vertical structure of the 3D mesh. These issues will be explored in future work.

Chapter 6

Applications of parallel mesh optimisation

Contents

The parallel mesh optimisation methods presented in Chapters 2-4 are of general applicability to unstructured mesh based FEM/FVM. The parallel mesh optimisation methods developed in this thesis will be demonstrated here through a series of transient incompressible CFD problems, namely: 3D flow past a cylinder; 3D flow past a sphere; and a differentially heated rotating annulus. These examples are chosen to illustrate the strengths and weaknesses of the method developed in this work.

The analysis presented in this chapter does not discuss at length the performance of the graph partitioner (ParMetis in this case) independently from the parallel mesh optimisa-

tion and dynamic load-balancing methods as this subject is explored extensively in the literature (e.g. [103, 61, 68, 67, 66, 69, 71]). In fact, the cost of graph repartitioning is included in the cost of data migration for all results presented (the joint operation usually referred to as data remapping). What will be examined separately however is the performance of data remapping and the essentially serial component of the algorithm - mesh optimisation. So that the computational costs associated with mesh optimisation can be viewed in perspective, the cost of the DDM is also reported. This cost is defined here to include the cost of matrix assembly, preconditioners and solvers. The experiments presented all highlight one attractive aspect of the algorithm used - namely that the overhead associated with parallel mesh optimisation over serial mesh optimisation contains within it the overhead associated with dynamic load-balancing, and this cost overhead is relatively low for all test cases considered.

Two different clusters were employed for the test problems presented here. The 3D flow past a cylinder and the differentially heated rotating annulus experiments were both run on a cluster of dual Intel® Xeon™ CPU 2.80GHz processors, networked with 1Gigabit Ethernet. The 3D flow past a sphere experiments were carried out on a cluster of dual Intel® Xeon™ CPU 2.40GHz processors, again networked with 1Gigabit Ethernet. Both of these clusters are dedicated systems employing PBS to manage the job queue, therefore there are only minor external factors which might effect the runtime of an experiment. Therefore all timings are taken from a single run unless otherwise stated. Timings were obtained through the use of tic/toc code, using *MPI_Wtime()* to obtain the wall clock time. All experiments were carried out using 64-bit floating point arithmetic.

6.1 Transient incompressible flow

The primitive variable form of the non-dimensionalised Navier-Stokes equations for an incompressible fluid is given by

$$\frac{D\boldsymbol{u}}{Dt} = \frac{1}{Re}\boldsymbol{\nabla}^2\boldsymbol{u} - \boldsymbol{\nabla}p, \tag{6.1}$$

$$\boldsymbol{\nabla} \cdot \boldsymbol{u} = 0, \tag{6.2}$$

where Re is the *Reynolds number*[1], p is the pressure, $\boldsymbol{u}$ the velocity containing three components and $D/Dt = \partial/\partial t + \boldsymbol{u} \cdot \boldsymbol{\nabla}$ is the total derivative. While heating effects are not considered for the cylinder and sphere experiments temperature, T, is still solved for as a passive scalar using the governing equation:

$$\frac{DT}{Dt} = \frac{1}{Pe}\nabla^2 T \tag{6.3}$$

where Pe is the *Peclet number*[2]. The use of such a passive scalar is useful when studying the mixing process in the wake of the cylinder.

The well known difficulty in satisfying the LBB stability condition [104], associated with mixed formulations is circumvented here by using a shared pressure/velocity mesh and introducing explicitly, a fourth order pressure stabilisation term into the continuity equation (6.2), see [105].

6.2 3D flow past a cylinder

In this section the flow past a cylinder is simulated with the aid of parallel mesh optimisation. Key points made in this section are: the load-balance, as measured by the number of solution nodes per partition, is good and validates the node prediction model (developed in Section 3.2) used to ascribe node weights before performing graph partitioning; and the graph partitioning remains well behaved. For this problem both the Reynolds number and the Peclet number have a value of 500, based on the maximum inlet velocity and diameter of the cylinder. The inlet (left hand side boundary of Figure 6.1) velocity has a linear

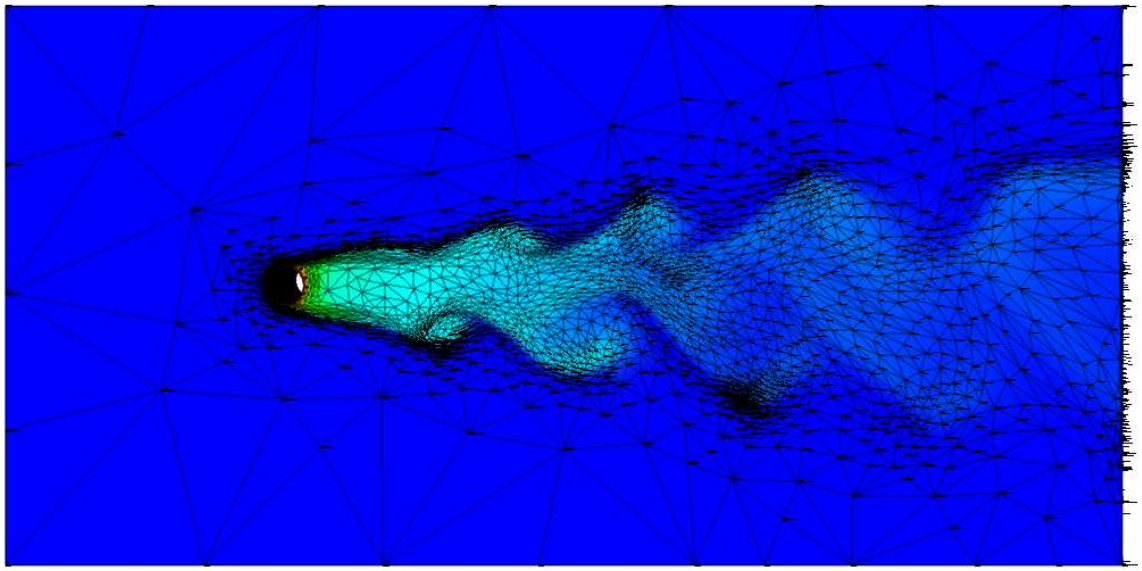
[1]The Reynolds number, Re, is a dimensionless parameter that indicates the dominance of inertial over viscous forces in a fluid.

[2]The Peclet number, Pe, is a dimensionless parameter that indicates the dominance of advection over conduction in the flux of heat in a fluid.

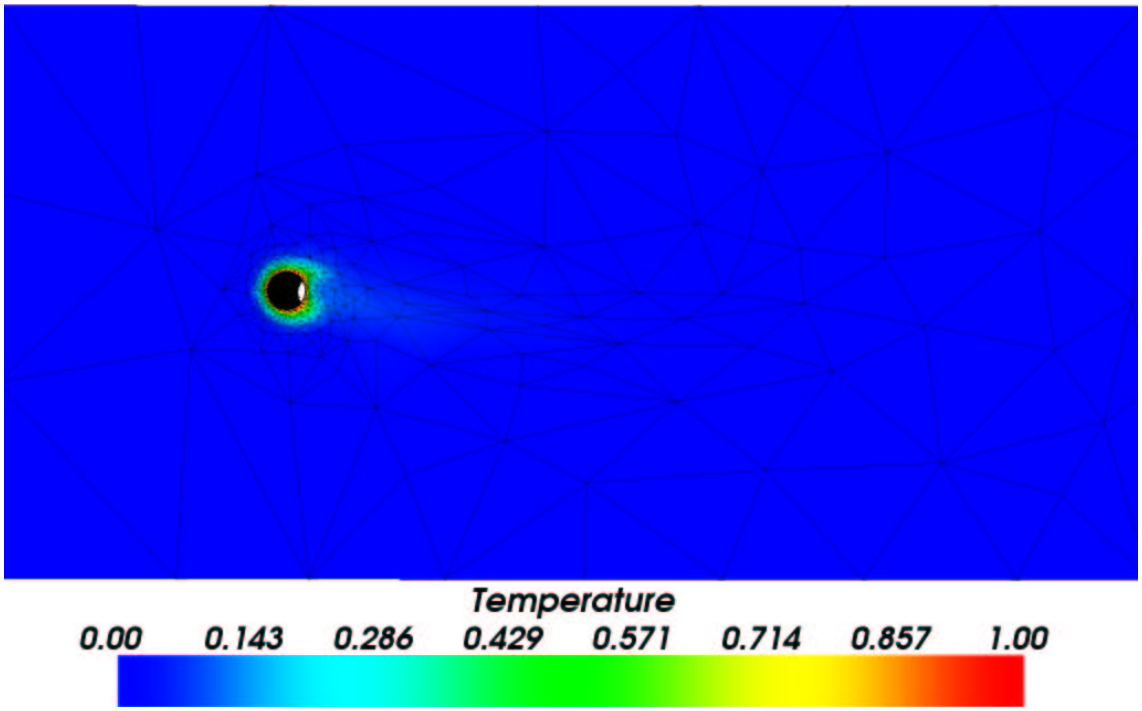
profile that is zero at the bottom and unity at the top while the velocity was not specified on the outlet, right. The mesh adapts every 100 time steps and the time step used is 0.005. The diameter of the cylinder is unity and the domain extent, shown in Figure 6.1, is 30 diameters long by 15 wide; the centre of the cylinder is placed 7.5 diameters from the inlet boundary on the left. The bottom of the domain and the surface of the cylinder have a a no-slip boundary condition applied (i.e. all velocity components are set to zero). A zero shear stress boundary condition was applied to the top and sides of the domain. The top and bottom boundaries of the three dimensional box shaped domain were 5 diameters apart. The interpolation error for temperature and the components of velocity was set to $\hat{\epsilon}_g = 0.02$ (see (2.6)). The minimum and maximum element lengths (h_{min} and h_{max}) were set to 0.005 and 3 diameters respectively. A typical domain decomposition is shown in Figure 6.2.

Figure 6.3 illustrates how the load-balance across eight domains and number of nodes in the halo (which is related to the edge-cut) varies over time. Other than the initial large increase in nodes at the start of the simulation, the number of nodes per domain remains steady. The size of the halo per domain is also well behaved. The relatively large size of the halo with respect to the size of the domain is due to this being a relatively small benchmark problem (only having about 30000 nodes in total) and both Halo-I and Halo-II nodes (see Figure 3.2) were required for this problem. It is also important to note that these graphs were partitioned using node weights predicted by (3.5) thus validating the node density prediction model employed in Section 3.2.1.

Figure 6.1 (a) shows the top of the computational domain where the temperature field is superimposed onto the computational mesh. Wedge glyphs indicate the direction of fluid flow. The von Kármán vortex street is immediately identifiable. The element sizes vary by approximately two orders of magnitude from large elements, in regions of the mesh outside the wake of the cylinder where there is little variation in the solution, to the very small in the wake and close to the cylinder where the solution varies rapidly. There are no velocity vectors in Figure 6.1 (b), because of the no slip boundary condition which is applied there. While there is no flow to resolve at the bottom of the domain, it is clear that the mesh has been optimised to capture the diffusion of the temperature field around the cylinder. The resulting velocity profile between these two layers is illustrated in Figure



(a) Top surface



(b) Bottom surface

Figure 6.1: The velocity, temperature and surface mesh (at time t=100). The top of the domain, (a), has a a slip boundary condition applied while the bottom of the domain, (b), has a no-slip boundary condition. In (a) it is clear that the mesh resolution is focusing in on the vortices of the von Kármán vortex street.

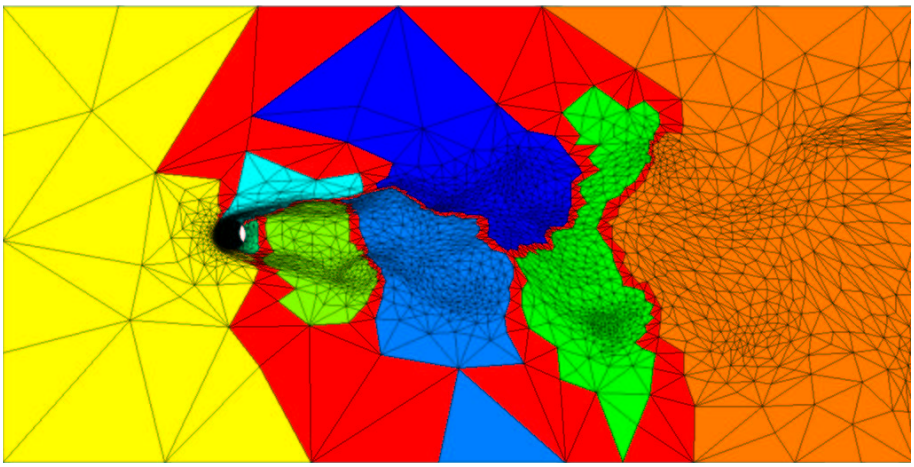


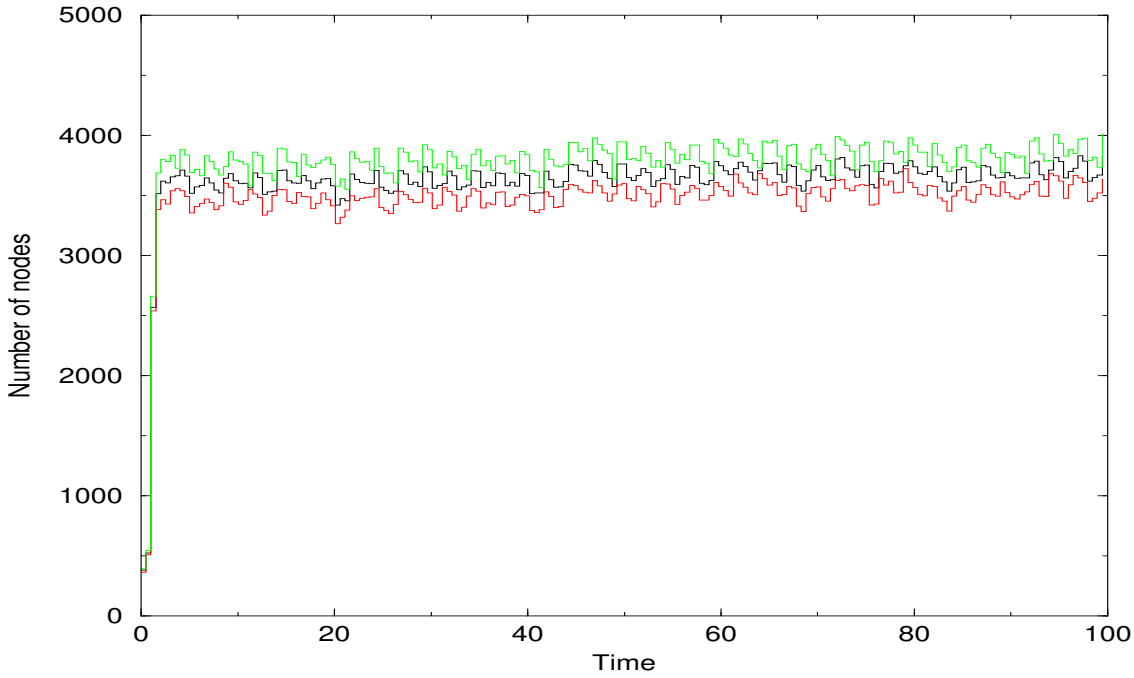
Figure 6.2: Graph partitioning for 8 domains at t=100, viewed from above.

6.4.

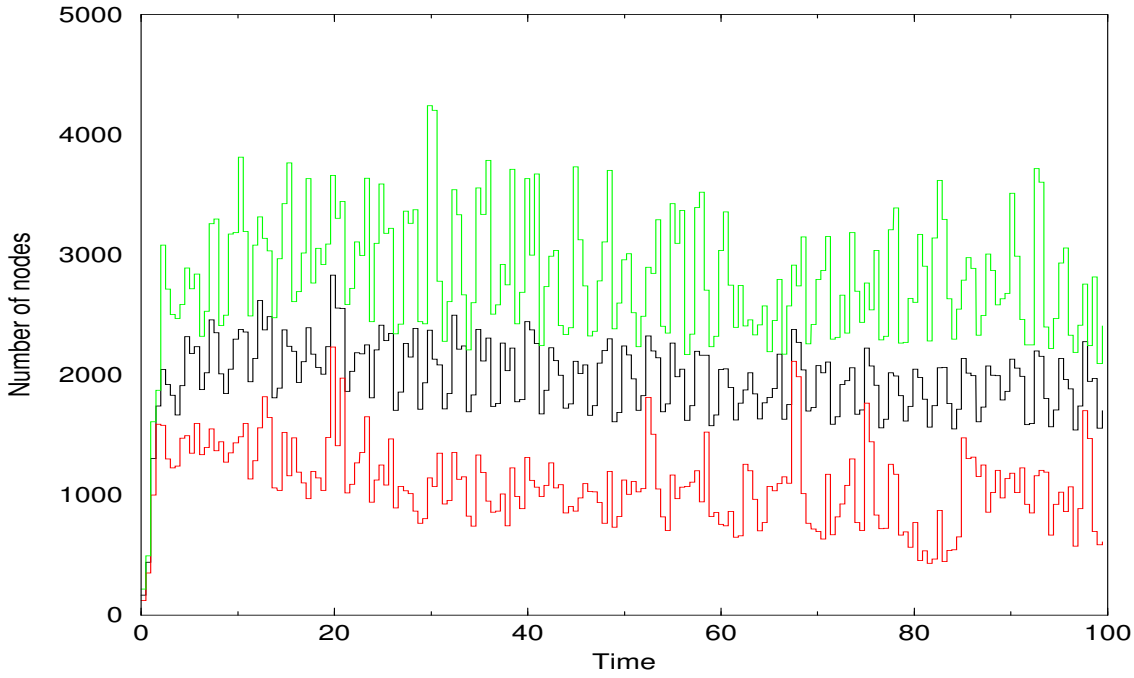
6.3 3D flow past a sphere

In this section the performance of the parallel mesh optimisation algorithm is deconstructed. The cost of dynamic-load balancing, which is also the parallel overhead associated with parallel mesh optimisation, is compared with the cost of applying mesh optimisation to a domain. These costs are relativised by comparing them with the cost of the DDM.

For this problem both the Reynolds number and the Peclet number have a value of 400, based on the maximum inlet velocity and diameter of the sphere. The inlet velocity is set to unity. The diameter of the sphere is unity and the domain extent is 10.5 diameters long by 5 wide; the centre of the cylinder is placed 2.5 diameters from the inlet boundary on the left. At the inlet boundary the velocity was set to unity in the direction normal to the boundary while the velocity was not specified on the outlet. The sphere has a no-slip boundary condition applied (i.e. all velocity components are set to zero). A zero shear stress boundary condition was applied to the sides of the domain. The opposing sides of the box are 5 diameters away from each other. The interpolation error was set to $\hat{\epsilon}_g = 0.03$ (see (2.6)). The minimum and maximum element lengths (h_{min} and h_{max}) were set to



(a) Limits on load-balance



(b) Halo extent

Figure 6.3: The top figure shows the variation in the maximum, average and minimum number of nodes per domain over the course of the simulation. The graph-partitioning was calculated using node weights as described in Section 3.2.1. The lower figure shows how the maximum, average and minimum number of nodes in the combined halo over all the domains varies over the course of the simulation.

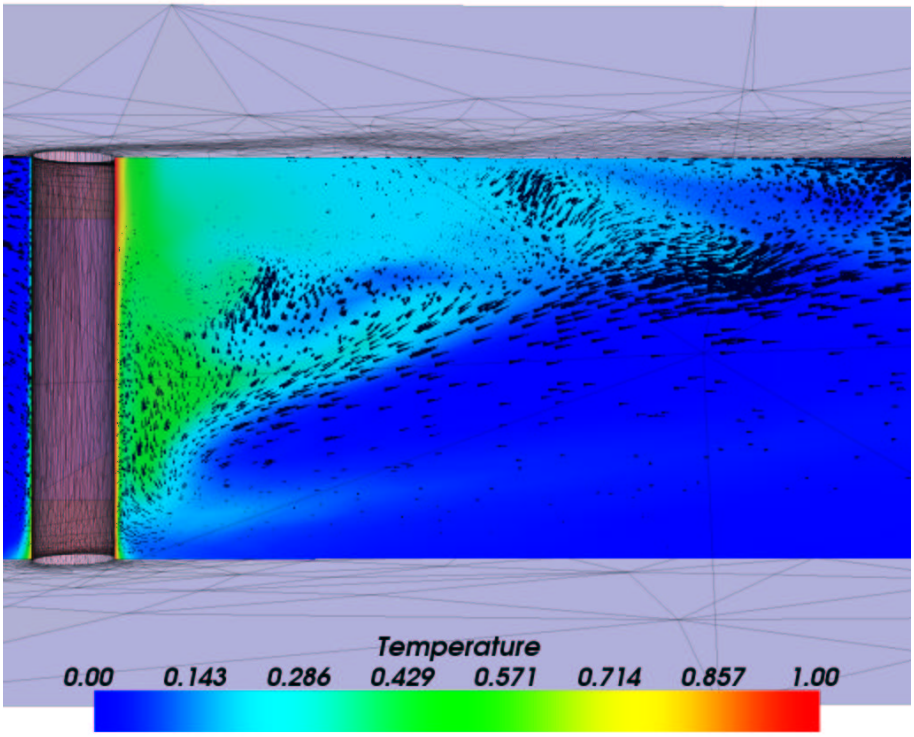


Figure 6.4: Cut plane though the centre of the domain showing velocity (wedge glyphs) and Temperature at t=100. Observe the thickness of the viscous boundary layer that has formed between the top and bottom surfaces. It is clear that the fluid flow is directed upward and mixing mostly takes place in the upper layers.

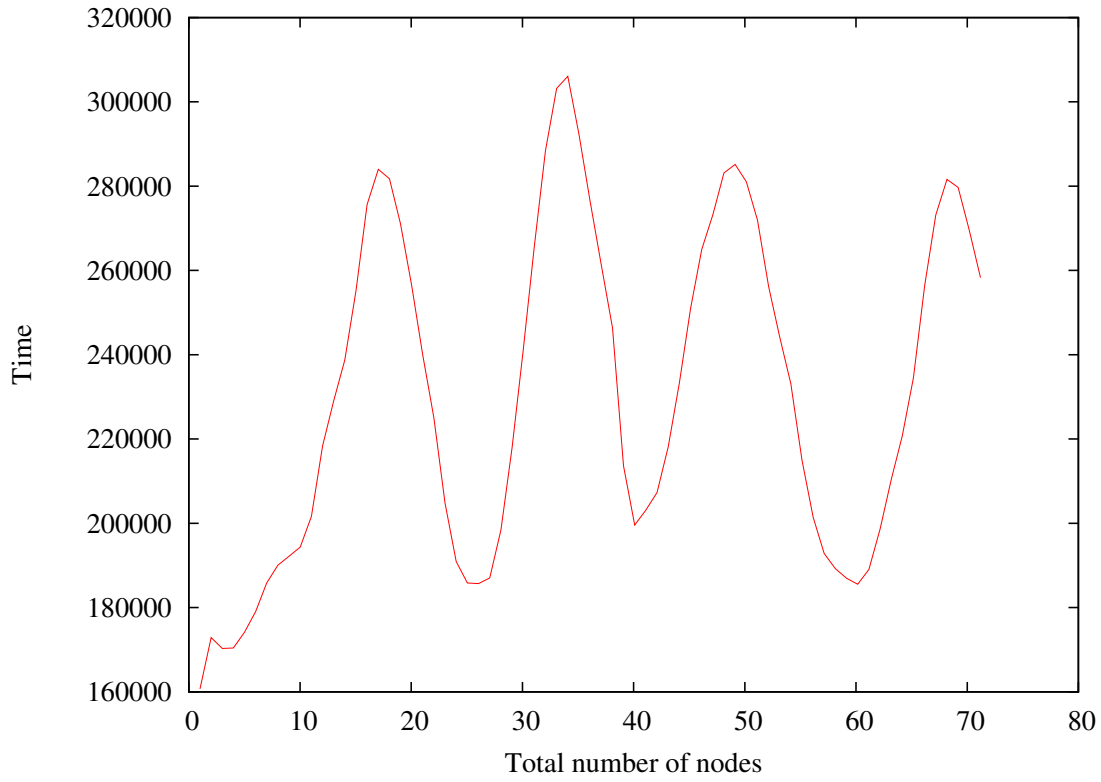


Figure 6.5: The total number of nodes in the mesh varies in time. Its periodic nature is related to the periodic shedding of eddies from the cylinder.

0.005 and 2 diameters respectively. Mesh optimisation is performed periodically. Figure 6.5 demonstrates how the total number of nodes varies in time through the application of mesh optimisation. One striking feature of this figure is the strong periodic variation of the total number of nodes which is related to vortex shedding. This again highlights the need for dynamic load-balancing for mesh optimisation methods as flow features such as vortices are followed through the domain by high densities of solution nodes.

The mesh adapts approximately every 0.125 time units where the initial time step is 0.025 and thereafter the time step is recalculated at the start of each time step using (2.24) with a maximum Courant number of 6.0. Figure 6.6 shows how the time step varies as the simulation progresses and is typical of how the time step behaves when mesh optimisation is used. In particular it is noted that it quickly settles down and in this example has time step values distributed around 0.008 time units.

Figures 6.9–6.12 show the cost of the DDM (meaning the cost of the parallel solvers),

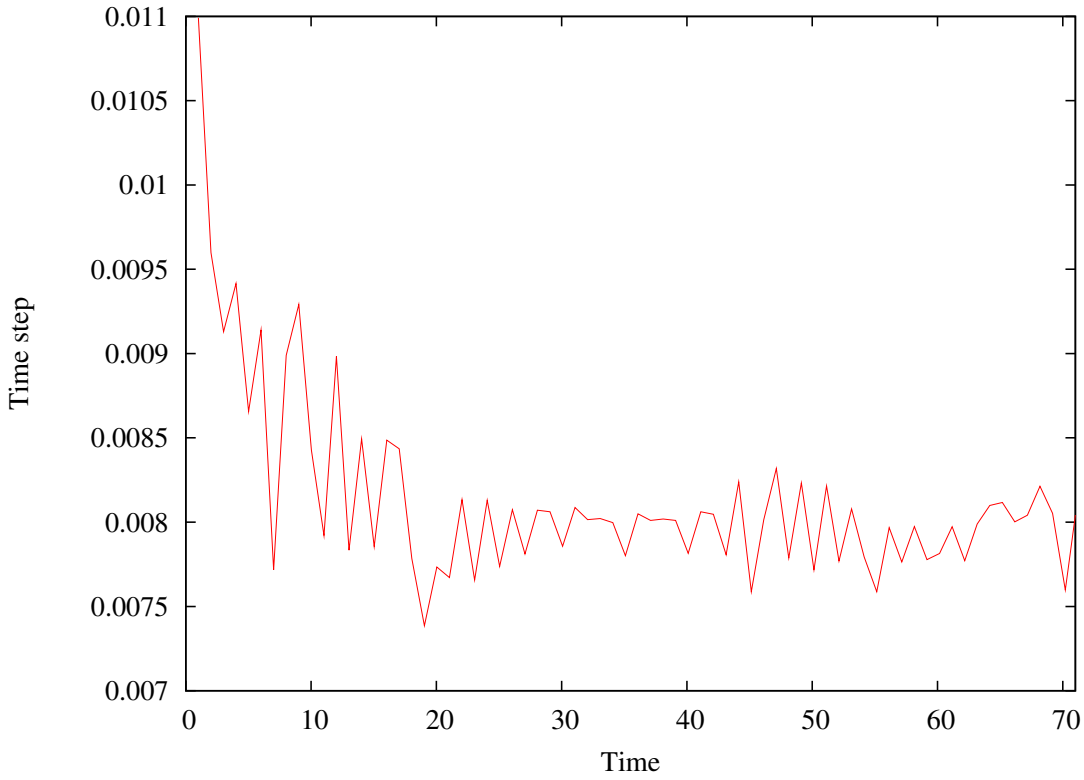
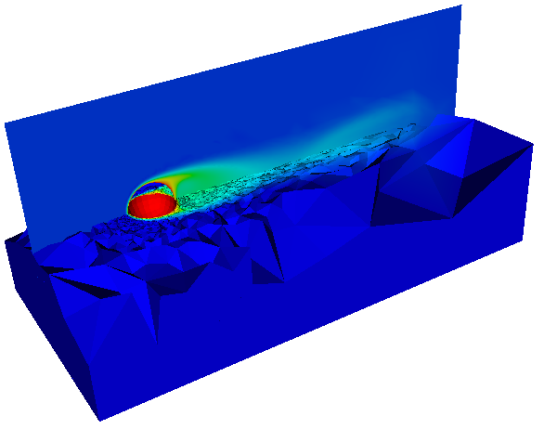
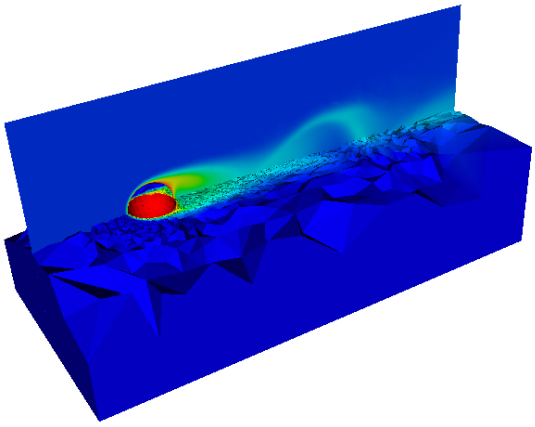


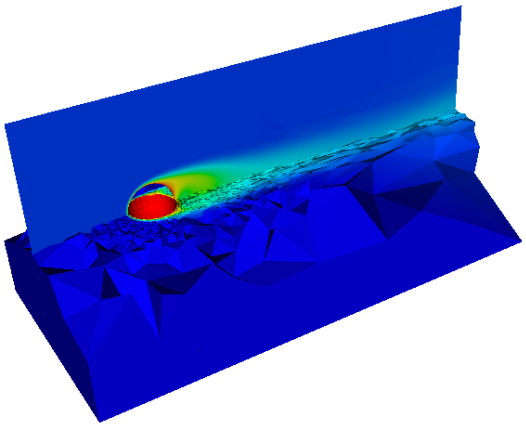
Figure 6.6: The time step, Δt, as a function of time, t, used for calculating flow past a sphere across four domains using mesh adaptivity.



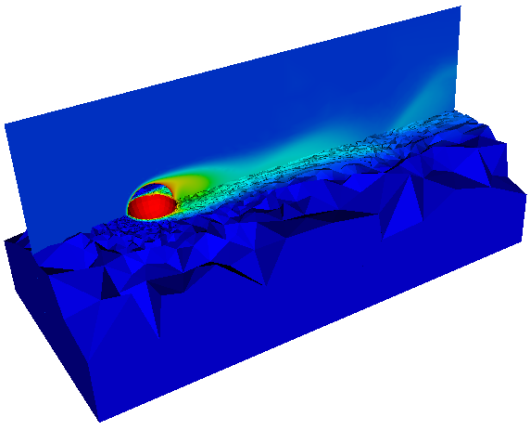
(a) 9.0 time units, 194347 nodes



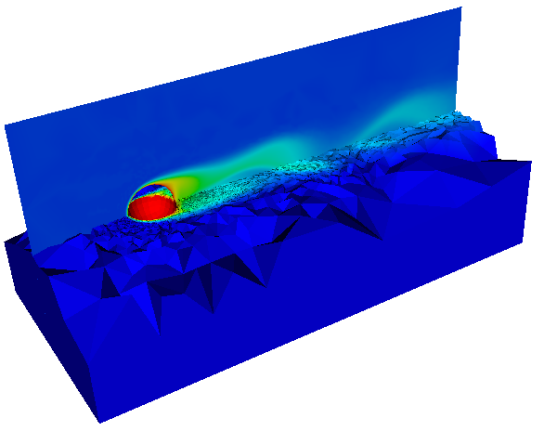
(b) 19.0 time units, 256421 nodes



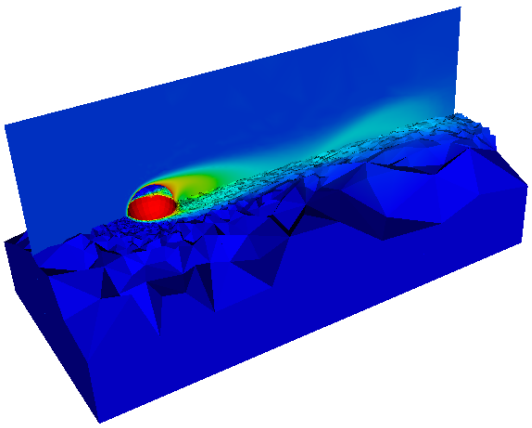
(c) 29.1 time units, 240875 nodes



(d) 39.1 time units, 199594 nodes



(e) 49.1 time units, 281087 nodes



(f) 59.2 time units, 185533 nodes

Figure 6.7: A series of snap shots of 3D flow past a sphere are shown. The upper section of the domain is each figure has been removed so that the mesh can be viewed and a vertical scalar cut plane is superimposed. Colour represents temperature.

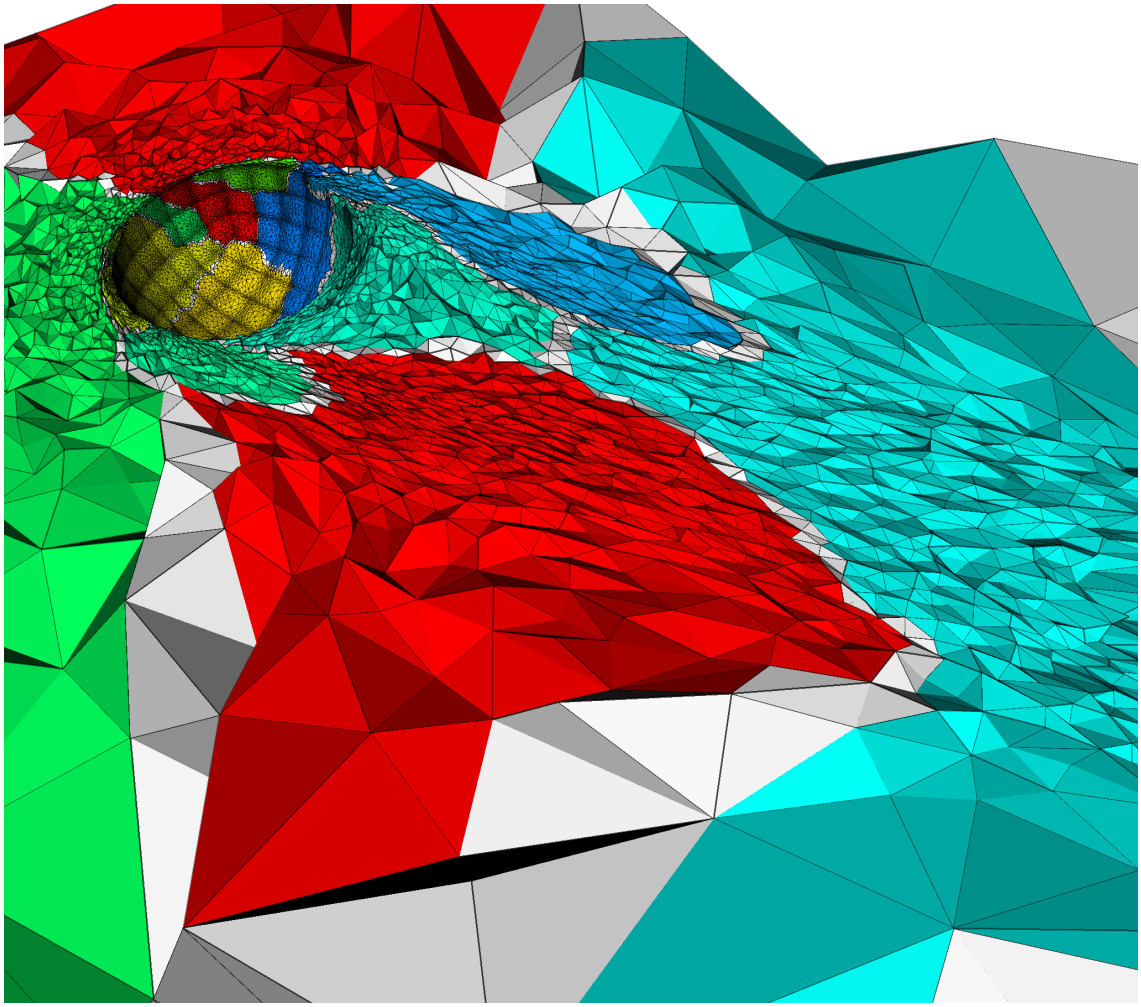


Figure 6.8: Typical graph partitioning of the 3D flow past a sphere problem. The upper half of the domain has been removed to expose the internal mesh. One can see that flow features are being resolved on the surface of the sphere highlighting the discrete surface representation used.

mesh adaptivity and data remapping for different numbers of domains as measured using wall clock time. The cost of the DDM is estimated as being half the total time spent in the application code before and after a parallel optimisation operation. The costs for mesh optimisation are the maximum, average and minimum time spent applying the serial mesh optimisation operation – this time excludes the waiting time between a process completing serial mesh optimisation and the last process to complete serial mesh optimisation. This allows us to clearly highlight the load imbalance occurring during mesh optimisation. The cost of data remapping is time lapsed before and after the data remapping operation.

Using the *eyeball norm*, the DDM and data remapping appear to scale well (this is quantified below), with the cost of data remapping being relatively low as one would hope. It is interesting to note how the costs of the three different operations are correlated with the total number of nodes in the domain, Figure 6.5. In particular, when the cost of mesh optimisation spikes, due to a jump in the number of degrees-of-freedom being added or removed from a partition, the cost of data remapping also peaks as more information must be migrated to balance the load.

However it is clear that there is a problem with the load-balance of the serial mesh optimisation operation applied across domains. The source of the problem lies in the fact that the focus of the load-balancing is only on balancing the work done by the DDM and not the work involved in mesh optimisation. As the cost of data remapping is relatively low, one obvious solution to this problem would be to load-balance specifically for mesh optimisation at the beginning of mesh optimisation and load-balance for the DDM after mesh optimisation. This requires a cost model for the mesh optimisation procedure so that an estimate can be made of the local cost of optimisation given the error metric and other local information such as the objective function. However a reliable cost model for mesh optimisation has not yet been developed. The key difficulty with developing a cost model for mesh optimisation is the complex nature of mesh optimisation algorithm. For example, there is only a very weak correlation between the objective function values (or other quantity derived from the error metric tensor) and the cost of mesh optimisation as a function of space. In this particular example the load-imbalance is exaggerated because only an isolated region requires significant optimisation and this region is distributed only over a small number of processors. This results in the majority of the optimisation work

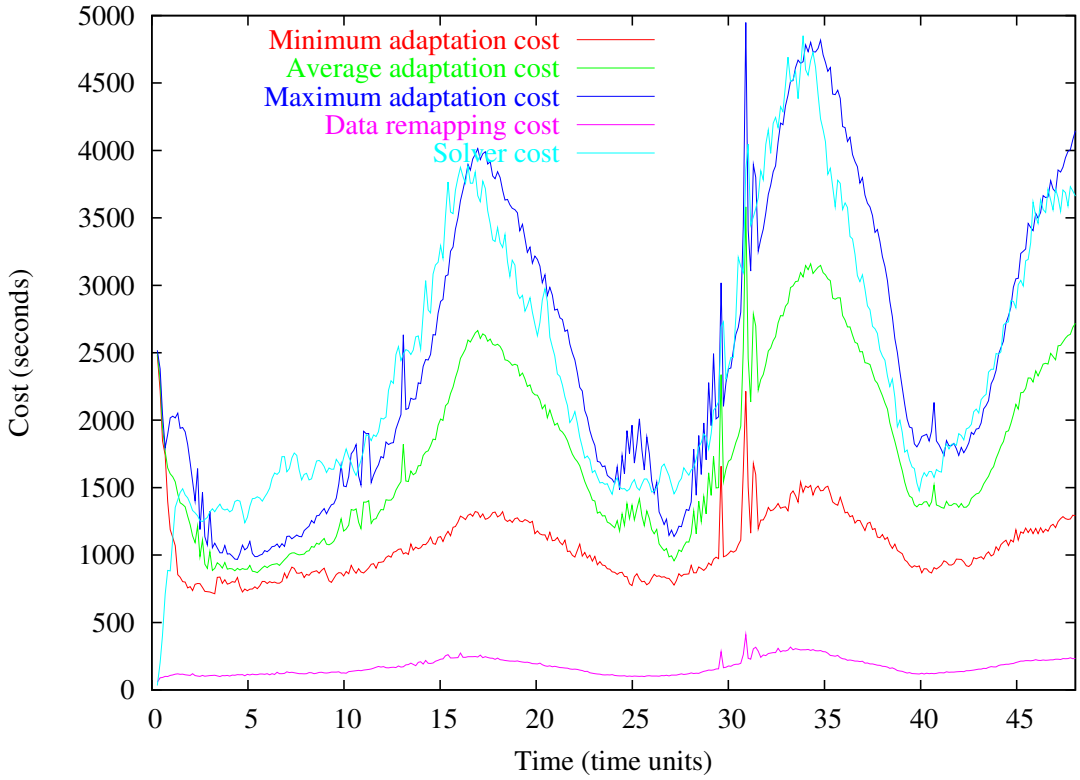


Figure 6.9: The computational cost of the DDM, mesh adaptivity and data remapping procedures for the flow past a sphere problem when using two partitions as measured using wall time.

being assigned to just a few processors.

Figures 6.13, 6.14 and 6.15 show how the cost of mesh optimisation, data remapping and the DDM vary respectively for 2, 4, 8 and 16 processors. Note that the peaks and troughs correlate with the number of nodes in the domain, Figure 6.5. To further illustrate the parallel performance of the method for this problem, three tables, containing cost and relative speedup data, are shown: Table 6.1 indicates the performance of the DDM; Table 6.2 indicates the performance of dynamic load-balancing and data remapping; and Table 6.3 shows the performance of serial mesh optimisation applied in parallel. The cost quoted in these tables is the average time measured for each operation within the $46th$ second time window. In each case the relative speedup figure is calculated by dividing the time taken to run on $2n$ processors by the time on n processors. The rational behind this untraditional definition is: a different preconditioner and DDM is used for the serial case and so it is difficult to give a useful comparison with the serial case; there is no data

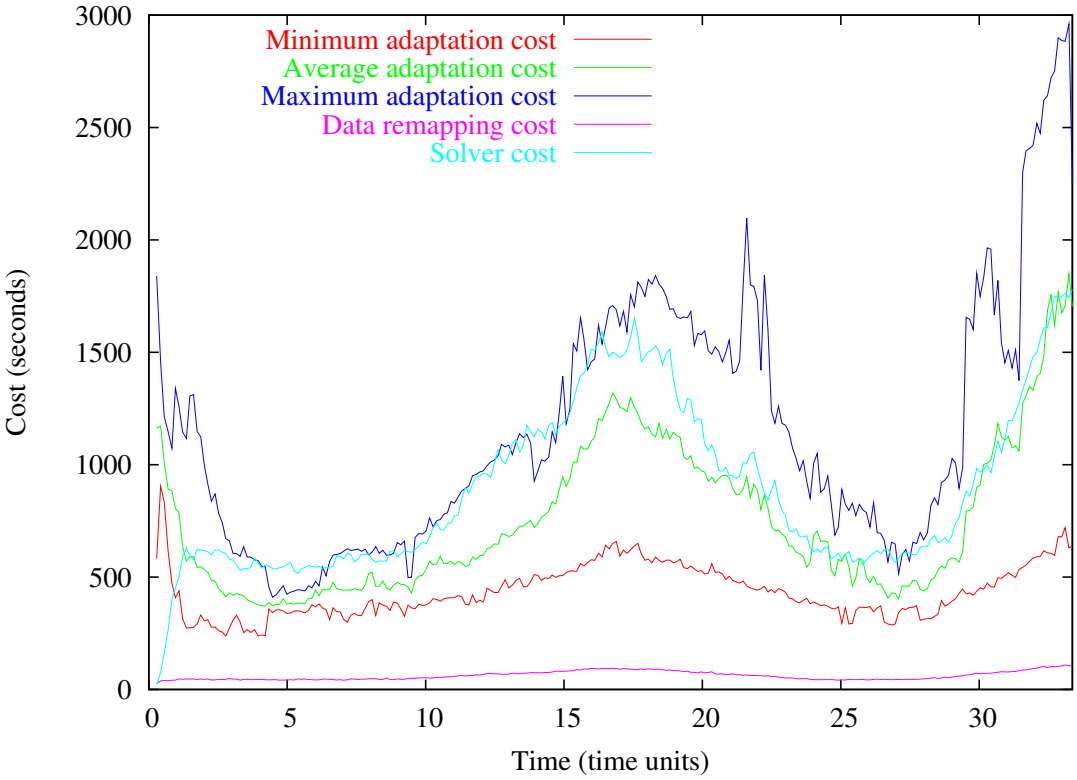


Figure 6.10: The computational cost of the DDM, mesh adaptivity and data remapping procedures for the flow past a sphere problem when using four partitions as measured using wall time.

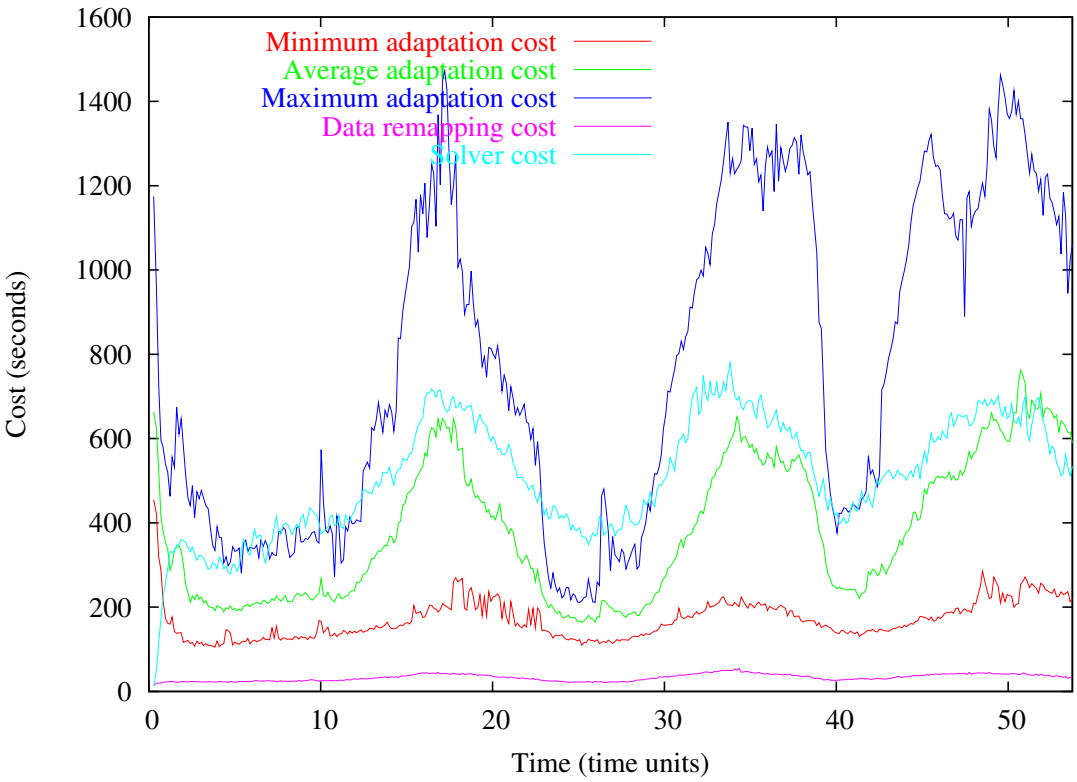


Figure 6.11: The computational cost of the DDM, mesh adaptivity and data remapping procedures for the flow past a sphere problem when using eight partitions as measured using wall time.

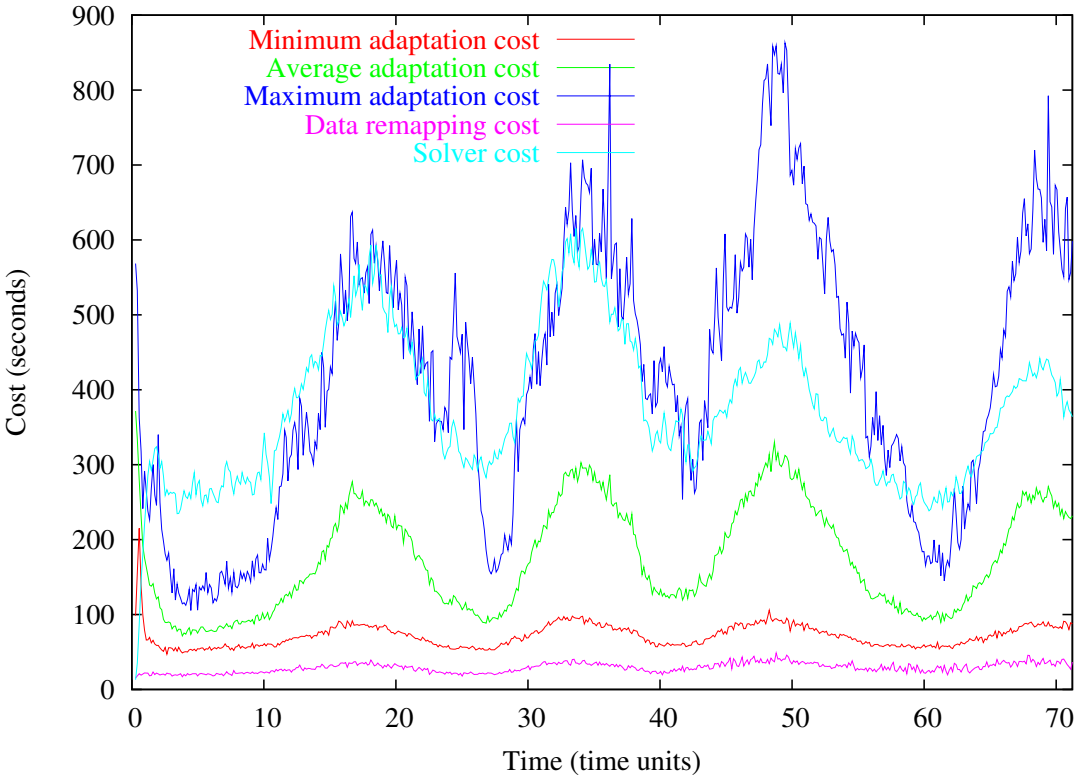


Figure 6.12: The computational cost of the DDM, mesh adaptivity and data remapping procedures for the flow past a sphere problem when using sixteen partitions as measured using wall time.

Number of processors	Averaged cost (time units)	Relative speedup
2	3585	-
4	1270	2.8
8	625	2.0
16	415	1.5

Table 6.1: Relative speedup of DDM

Number of processors	Averaged cost (time units)	Relative speedup
2	220	-
4	83	2.7
8	46	1.8
16	37	1.2

Table 6.2: Relative speedup of data remapping

remapping performed in serial so a comparison of that type cannot be made; this form yields more information about the scalability of the method. There are a few aspects to these tables that require some consideration. The first is the fact that some super-linear speedup figures are quoted. There are two factors which contribute to these high values: the speedup is not based on a serial benchmark and thus are going to appear higher than what might otherwise be expected; it is observed that two serial processes will run slower on a dual Xeon than a single process [106]. Dual Xeons, such as those used in these tests, share a common front-side bus. Many experiments, such as those published by Guiang et al. [106], show that applications that have less FLOPS per memory reference (e.g. sparse matrix multiplications) can perform poorly on dual Xeons due to limitations of the shared bus memory architecture. Thus, it is postulated that this problem is alleviated as the size of the problem per processor decreases. The final, rather conspicuous, performance metric that requires explanation is the relative speedup of mesh optimisation going from 8 to 16 processors, Table 6.3. This is interpreted as an artifact of the load-imbalance in mesh optimisation as previously discussed, where the mesh regions which require most optimisation become proportionately much better distributed.

Number of processors	Averaged cost (time units)	Relative speedup
2	1204	-
4	555	2.2
8	670	0.8
16	129	5.1

Table 6.3: Relative speedup of mesh optimisation

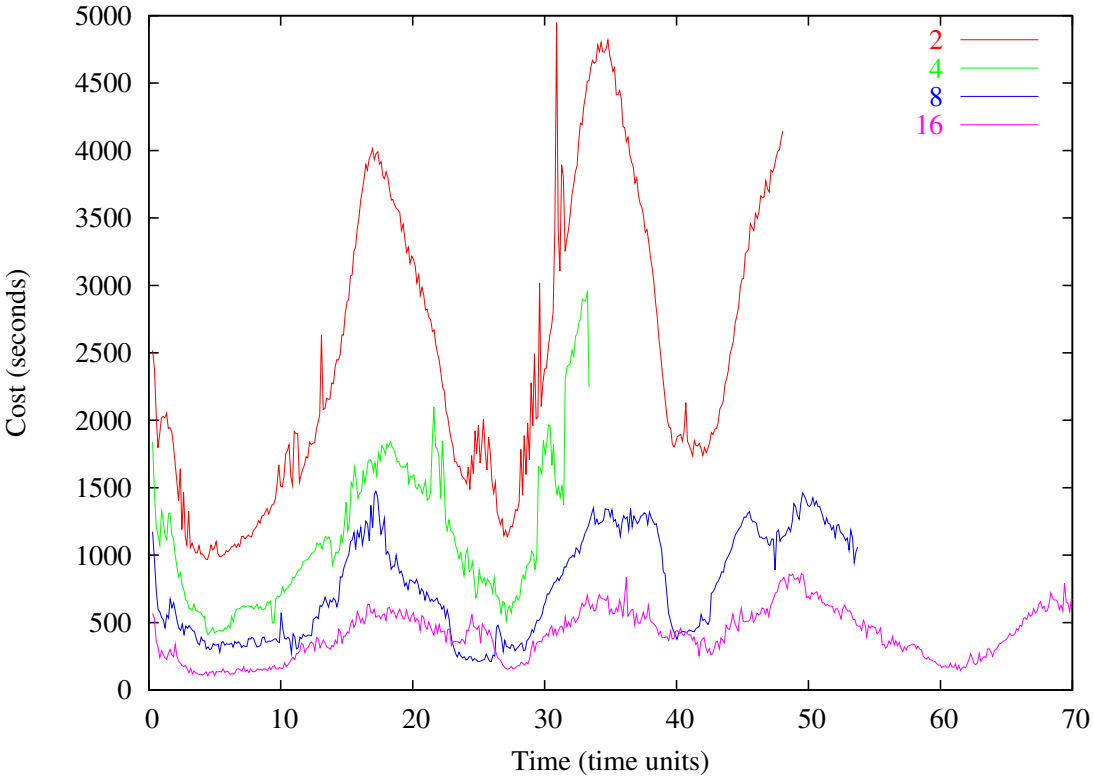


Figure 6.13: The cost of serial mesh optimisation on 2, 4, 8 and 16 processors for the 3D flow past a sphere problem.

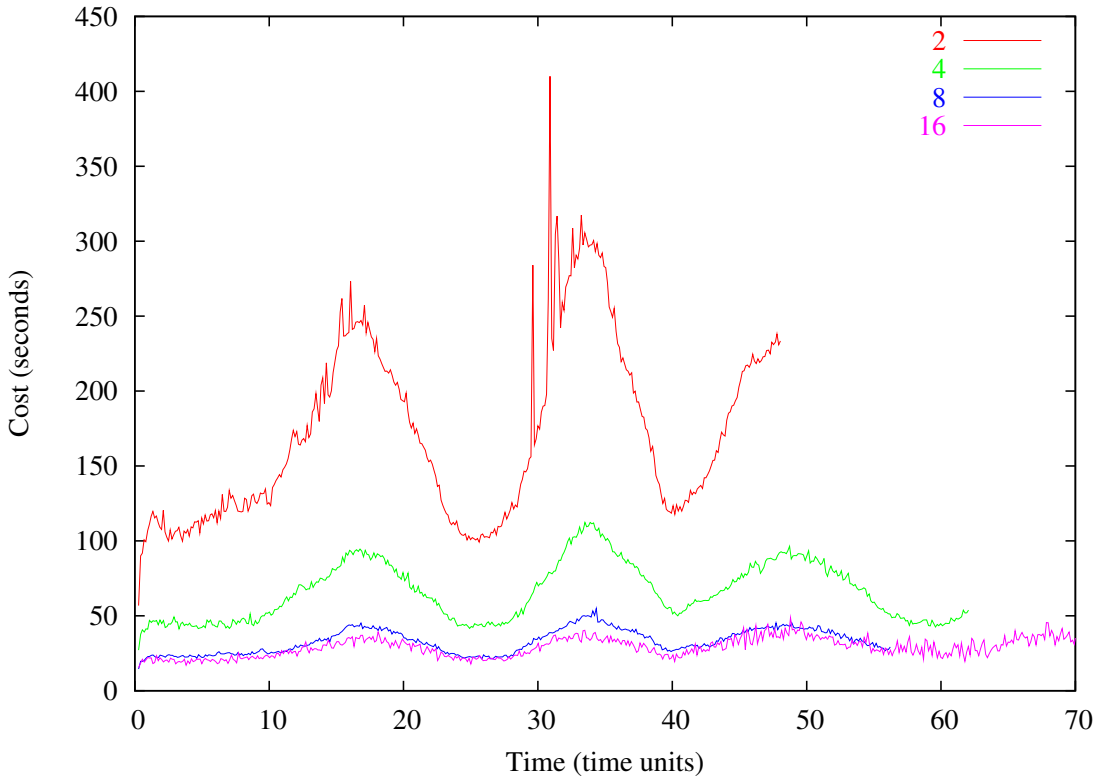


Figure 6.14: The cost of data remapping on 2, 4, 8 and 16 processors for the 3D flow past a sphere problem.

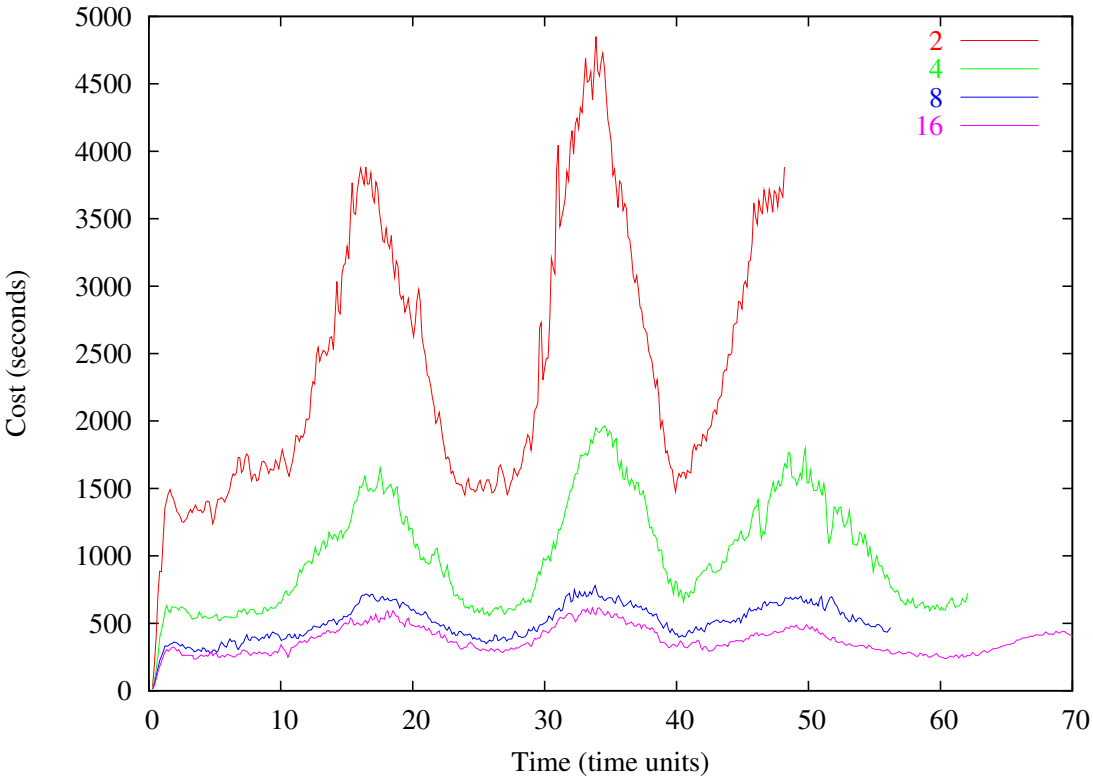


Figure 6.15: The cost of solver on 2, 4, 8 and 16 processors for the 3D flow past a sphere problem.

6.4 Differentially heated rotating annulus

Many laboratory experiments of the motion of a fluid in an annulus which rotates about a vertical axis and is subject to a horizontal temperature gradient have been presented in the literature (for a review see [107]). These experiments show that when the rotation rate exceeds a critical level (which depends on many parameters), centrifugal forces inhibit overturning motion and promote a different flow structure, which is often termed *sloping convection* or *baroclinic waves*. This motion is generally non-antisymmetric and mostly confined to meandering *jet streams*.

The primitive variable form of the non-dimensionalised Navier-Stokes equation is given by

$$\rho \frac{D\boldsymbol{u}}{Dt} + \boldsymbol{F_c} = \rho\nu\boldsymbol{\nabla}^2\boldsymbol{u} - \boldsymbol{\nabla}p + \boldsymbol{F_b}, \tag{6.4}$$

where ρ is the density, ν is the kinematic visocity $\boldsymbol{F_c}$ is the Coriolis body force (as the system is solved in a rotating frame of reference where the annulus appears stationary),

$$\boldsymbol{F_c} = 2\rho\boldsymbol{\Omega} \times \boldsymbol{u}, \tag{6.5}$$

where $\boldsymbol{\Omega}$ is the angular velocity. $\boldsymbol{F_b}$ is the buoyancy body force,

$$\boldsymbol{F_b} = -\rho\boldsymbol{g} \tag{6.6}$$

where $\boldsymbol{g}$ is the acceleration due to gravity. A linear equation of state is used for density,

$$\rho = \rho_0(1 - \alpha(T - T_0)) \tag{6.7}$$

where α is the coefficient of thermal expansion, ρ_0 is the density of the fluid at temperature T_0 and T is the temperature of the fluid.

In the experiment considered here a fixed lid annulus is set up with an internal radius of $4cm$, an outside radius of $8.64cm$ and a height of $13.5cm$. The temperature difference between the inside and outside of the annulus is $10K$, where the outside is the hotter. A no slip boundary condition is applied to the sides and bottom while a free slip bound-

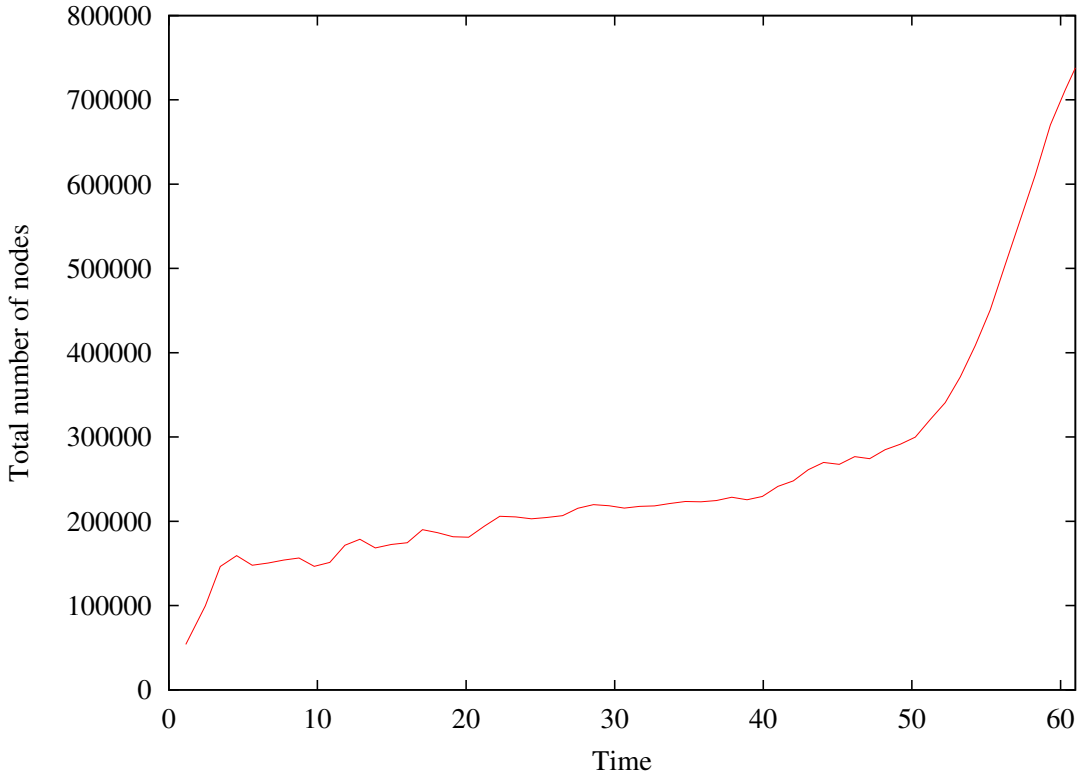


Figure 6.16: The total number of nodes in the mesh varies in time. The steep increase in the number of nodes can be understood with the aid of Figure 6.18 where one can see that there is a *spin-up* time before baroclinic waves become fully established.

ary condition with zero normal flow is applied to the top. The annulus is rotating about it's axis of symmetry at $2rads^{-1}$. Other physical quantities are: kinematic viscosity, $\upsilon = 0.025cm^2s^{-1}$; thermal conductivity, $\kappa = 0.001Wcm^{-1}K^{-1}$; the density of the fluid at T_0 (the temperature of the inner boundary), $\rho_0 = 10^{-3}Kgcm^{-3}$; and the coefficient of thermal expansion, $\gamma = 2 \times 10^{-4}K^{-1}$. The interpolation errors are: $0.1cms^{-1}$ for velocities in the horizontal; $0.025cms^{-1}$ for the vertical velocity; and $0.1K$ for temperature. The minimum and maximum element lengths (h_{min} and h_{max}) are set to $0.025cm$ and $5cm$ respectively. The total number of solution nodes and the time steps used are illustrated in Figures 6.16 and 6.17 respectively.

The set of frames in Figure 6.18 show the flow spinning up and meandering jets or waves forming. It is clear that as the flow becomes more complex, additional resolution is required as is illustrated in Figure 6.16. Figure 6.19 slows the detailed features of the temperature field and the corresponding graph-partitioning for this experiment on eight

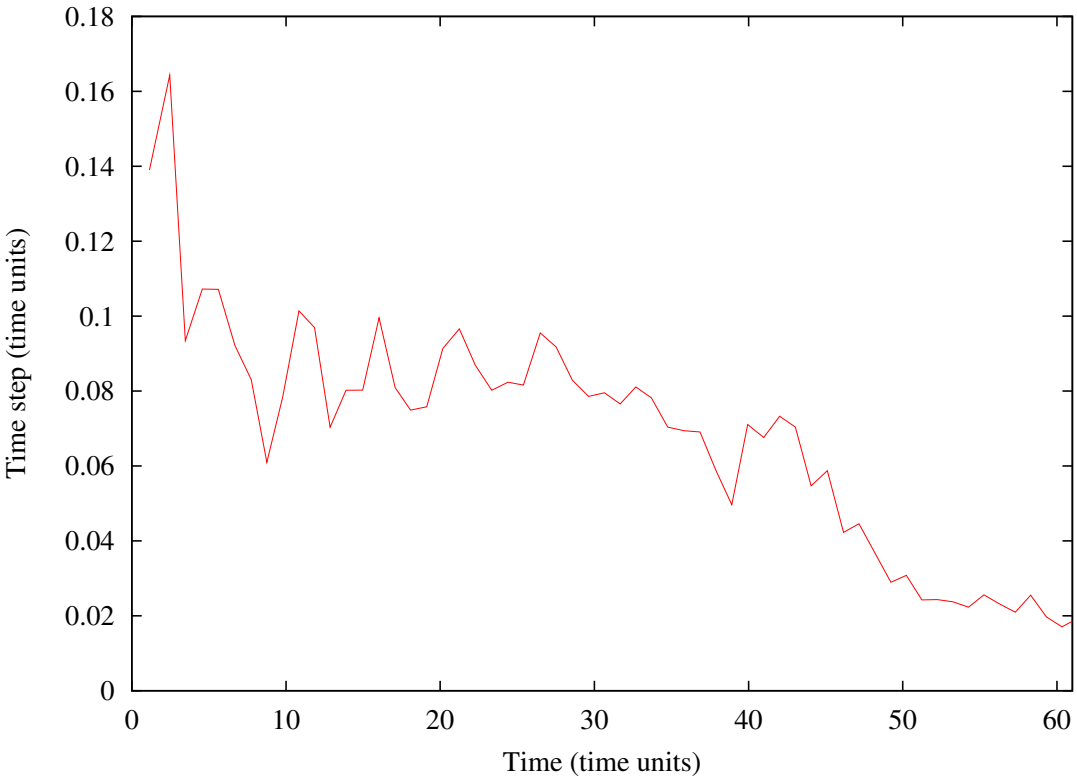


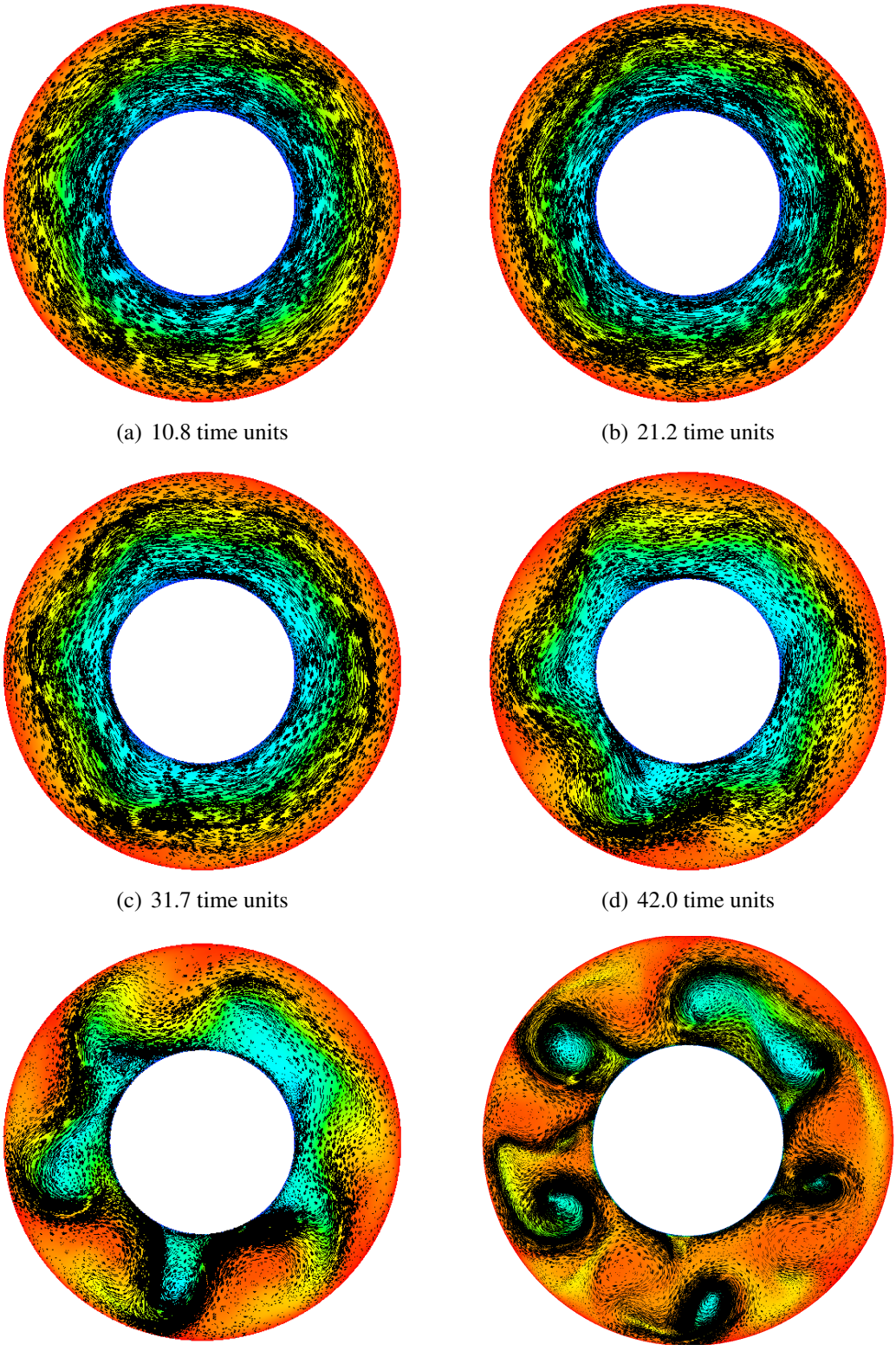
Figure 6.17: The time step, Δt, as a function of time, t, used for the differentially heated rotating annulus experiment.

processors is shown in Figure 6.20. In striking contrast to the previous experiment, because gradients are more evenly distributed throughout the domain, the load-balance of mesh optimisation is good (see Figure 6.21). In addition, mesh optimisation is now only applied every time unit in experiment time (this is roughly every 20 time steps) which is more than sufficient for this experiment. This results in a more favourable comparison between the costs of the DDM, mesh optimisation and data remapping, see Figure 6.22. In particular, note that the cost of data remapping becomes negligible when considered with the other costs. This experiment was also carried on sixteen processors and a relative speedup of 2 was observed. Again, the likely explanation for this high value is the degraded performance of memory intensive processes on dual Xeons as discussed in the previous section.

6.5 Conclusions

This chapter demonstrated the capabilities of parallel mesh optimisation. In problems where variations in the solution gradients are localised within the domain load-imbalances are experienced within the parallel mesh optimisation. To address this problem a cost model will need to be developed for the mesh optimisation process. This may take the form of an analytical model or may require an empirical model such as an artificial neural network. However, in practise the performance scalability of these problems can be greatly improved by one, or a combination of a number of, methods: smoothing the metric so that mesh optimisation is less sensitive to changes in gradients; by increasing marginally the accuracy sought by the mesh optimisation method beyond that required by the solution and performing mesh optimisation less frequently.

Fortunately, most practical problems of interest are more complex (e.g. ocean modelling [108]) and the variation of solution gradients tends to be more evenly distributed throughout the domain, as in the case of the spinning annulus in Section 6.4. Under such conditions the parallel mesh optimisation method performs very well. In particular the parallel overheads associated with applying mesh optimisation in parallel and performing dynamic load-balancing appear minimal. Although more experiments on greater numbers



(a) 10.8 time units (b) 21.2 time units

(c) 31.7 time units (d) 42.0 time units

(e) 52.2 time units (f) 62.3 time units

Figure 6.18: A series of snap shots of 3D flow in a differentially heated rotating annulus. Colour represents temperature.

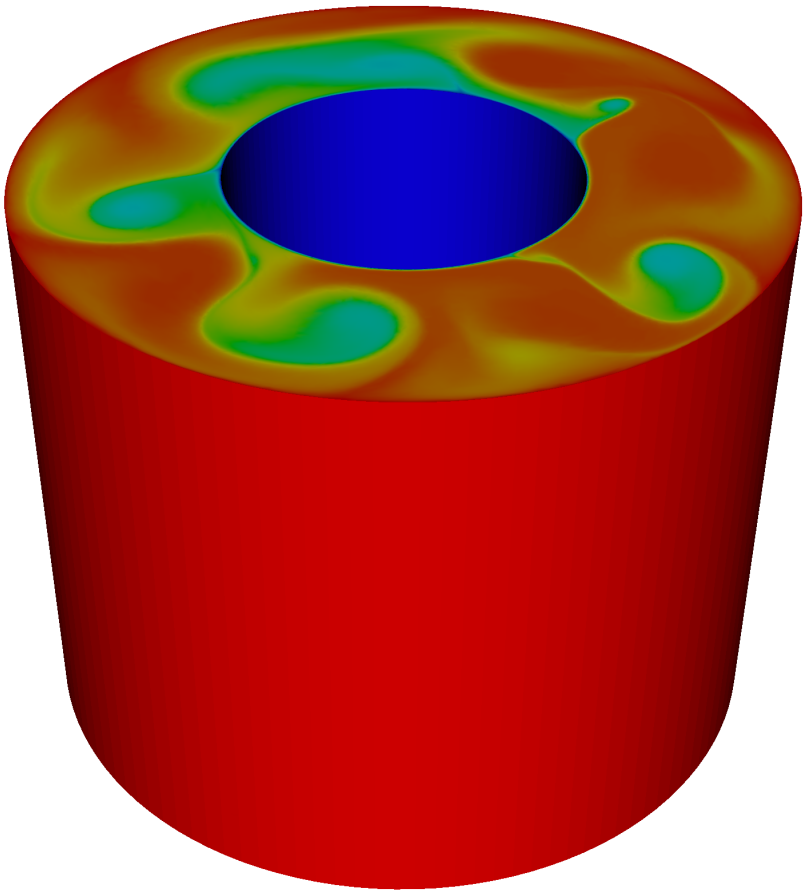


Figure 6.19: Temperature profile looking on top of the spinning annulus at 56.3 time units.

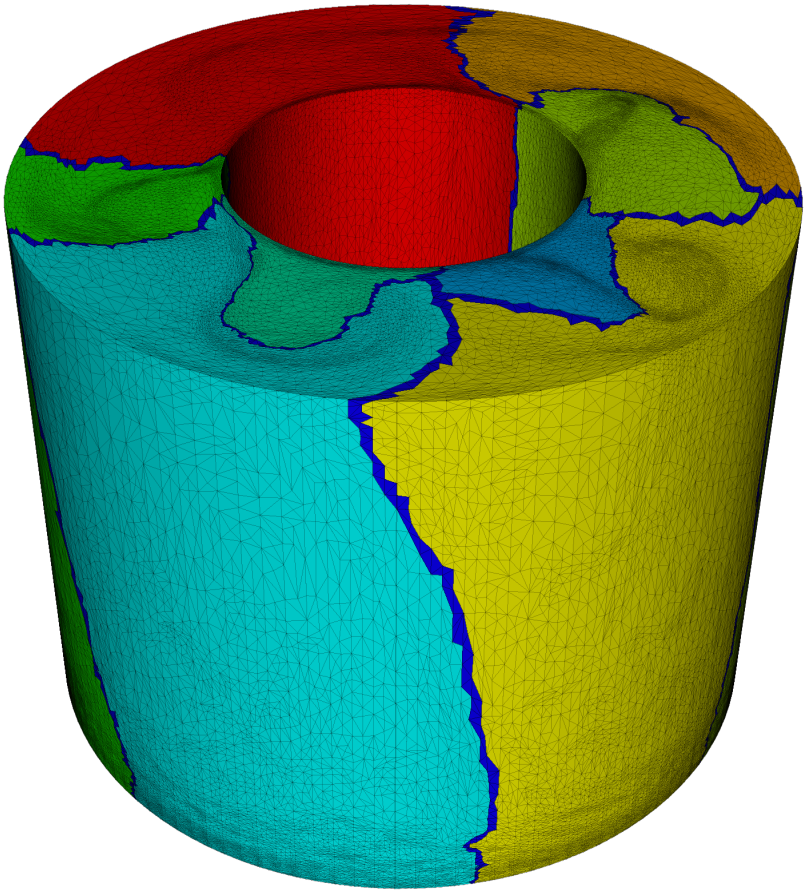


Figure 6.20: Typical graph partitioning for differentially heated rotating annulus problem (56.3 time units).

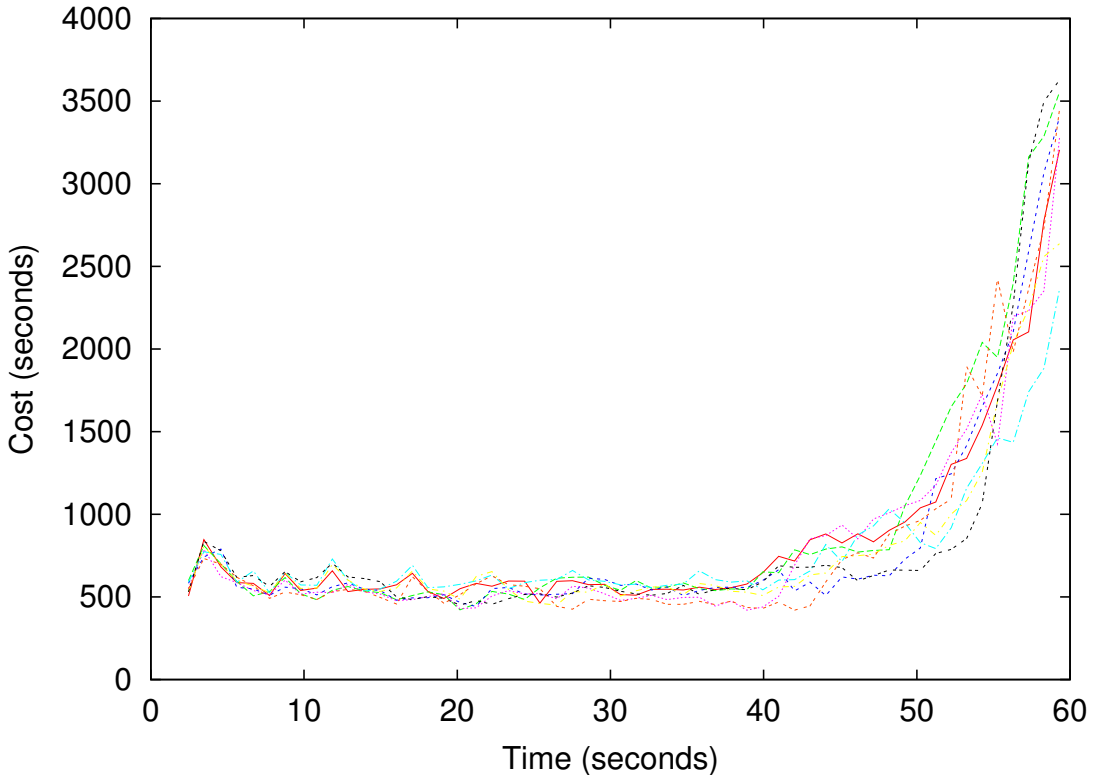


Figure 6.21: The computational cost of mesh optimisation for each of the eight sub-domains.

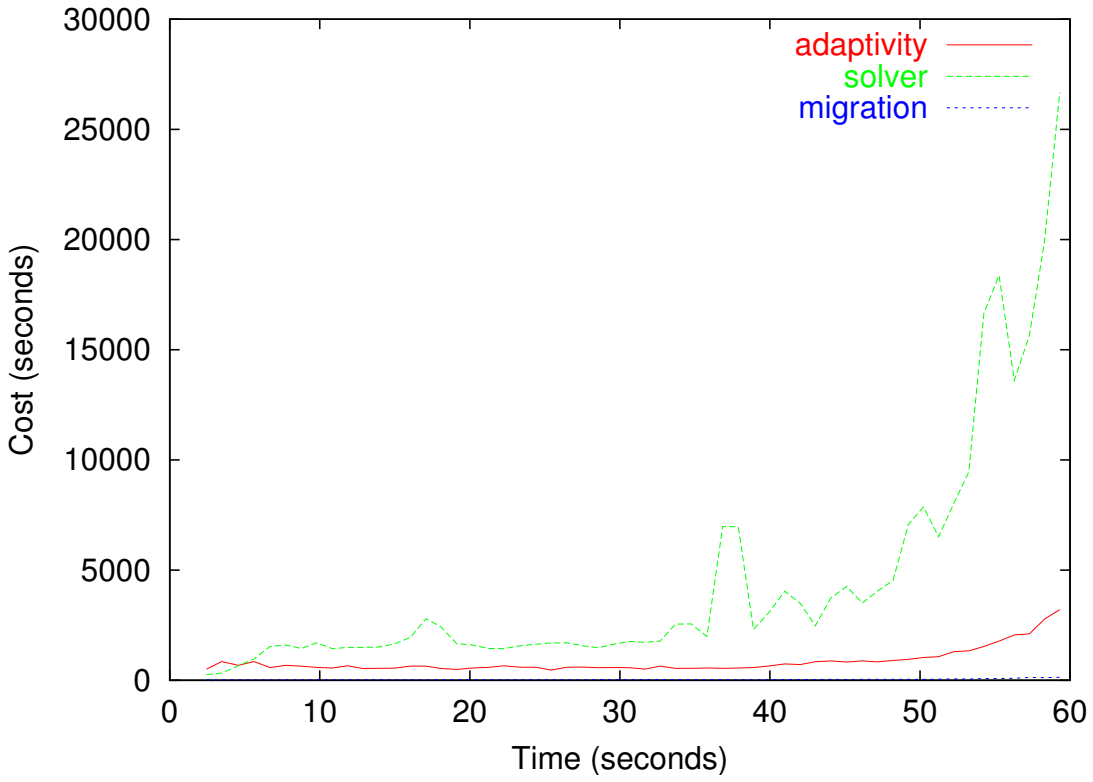


Figure 6.22: The relative computational costs of the DDM, mesh optimisation and data remapping are shown. The graph showing the cost of data remapping is obscured by the bottom horizontal axis.

of processors and on different architectures would be necessary to make decisive statements about scalability [3], these experiments do indicate that it is reasonable to expect good performance results for a moderately sized Beowulf cluster. Further experiments will be required to verify and extend this extrapolation.

[3] *Unfortunately, the performance of a code on a scaled down system is often not very impressive. There is a straightforward solution to this dilemma — project your performance results linearly to a full system, and quote the projected results, without justifying the linear scaling. Be very careful not to mention this projection, however, since it could seriously undermine your performance claims for the audience to realise that you did not actually obtain your results on real full-scale hardware.* (Quote from *Twelve ways to fool the masses when giving performance results on parallel computers* [109].)

Chapter 7

Conclusion

Contents

The previous chapter demonstrates that the parallel mesh optimisation method developed can be used effectively for parallel FEM applications. It is clear from the tests conducted

that the overheads associated with dynamic load-balancing (i.e. repartitioning and data migration costs) are minor when compared with the costs of serial mesh optimisation and the solver (CG and GMRES with a SSOR preconditioner). This is particularly significant as this cost is one and the same as the overhead incurred by performing mesh optimisation in parallel.

The results also highlight the fact that no effort has been made toward load-balancing the mesh optimisation as distinct from load-balancing the solver. In the case of flow past a sphere for example, Figures 6.9 to 6.12 show a wide variation in the cost of mesh optimisation across processors. However this should be viewed as a worst case scenario due to the fact that mesh optimisation was performed at each time step and the regions with fast changing curvature (thus regularly requiring mesh optimisation) are localised and by implication, so is the work to be done by the mesh optimisation procedure. For most practical applications, regions of rapidly varying curvature are more evenly distributed and usually multiple time steps are solved between applications of mesh optimisation. This lowers the actual load-imbalance and reduces the the cost of mesh optimisation with respect to the solver, thus reducing the impact of load-imbalances in mesh optimisation. However, these are evidently issues that require further investigation.

In this chapter the strengths and possible weaknesses of the implemented parallel mesh optimisation will be examined and possible solutions discussed. The chapter will conclude with a look at likely trends in mesh optimisation in the near future. This will focus on the next generation of mesh optimisers which aim to adapt general polyhedral meshes achieving high fidelity hex-dominant meshes and moving mesh methods.

7.1 Dynamic load-balancing

Apart from the solver itself, load-balancing is arguably the most important issue in parallel computing, especially dynamic load-balancing when the load is variable or unpredictable as is the case when adaptive computational methods are employed. Dynamic load-balance has three basic requirements: a means by which to estimate the work to be done; a method

to partition the work among a given set of processors; a method to distribute the work among the available processors.

7.1.1 Estimating the cost of computation: mesh optimisation cost model

In terms of cost models, load-balancing the load for the solver is usually relatively straightforward when it is assumed that poorly conditioned elements and element CFL numbers, when an adaptive explicit time stepping is used [110], are equidistributed. In this case the load is well balanced if the number of solution nodes is approximately equal on each subdomain and the communication cost between these partitions is balanced. As was seen in the previous chapter the cost of data migration is relatively low, with respect to the cost of the solver and serial mesh optimisation, and so its cost can generally be neglected.

The challenge facing the current mesh optimisation method is to develop a cost model to facilitate load-balancing of mesh optimisation. A cost model is critical because of the relatively high cost of the method for some applications. Once such a model is available, load-balancing can be achieved by either: explicitly calculating a separate partitioning for the solver and for the mesh optimisation step, remapping data as necessary; or solving a multi-objective graph partitioning problem seeking to balance both the cost of the solver and mesh optimisation together.

The development of such a model however is not without its difficulties. The key issue is that there is only a weak correlation between the error metric (and its derived quantities such as the optimisation objective function) and the actual amount of work that must be done on a sub-mesh before some quality tolerance has been achieved. For example, one might suggest a simple linear model relating the local change in the number of elements (see Section 3.2.1) to the cost of optimisation. While there are a few problems with this model, perhaps the most obvious relates to the fact that anisotropy in the mesh is a desired feature when anisotropy exists in the solution. Thus a significant level of work in mesh optimisation may involve changes to the shape of the mesh rather than the density of elements or nodes. There is also the fact that the relation between the change in element

or node density and the error metric is highly non-linear. Similar problems relate to use of the objective function or other quality measures.

The hope is that a systematic study of the individual contributions of different adaptation methods (i.e. edge collapsing/splitting, face/edge swapping and mesh smoothing) to mesh properties such as node density, element shape and mesh orientation (all of which can be clearly related to the error metric [9, 10]) will cast light on how such a cost model may be devised. Such a study would take a similar form to the study by Freitag et al. [111, 8, 112], but with an increased emphasis on modelling the cost benefit. Indeed, it may be that the optimisation method itself may need to be modified to facilitate a cost model being devised.

7.1.2 Partitioning

One novel aspect of the work presented here is the application of edge weights to discourage the partitioning of mesh regions that may require further optimisation. Avoiding such regions reduces the overall communication overhead associated with parallel mesh optimisation while still load-balancing. This approach is both simple and powerful. It can be applied for similar effect to other approaches to parallel mesh optimisation methods, e.g. Freitag et al. [49] and Jimack [32]. Further work is required to assess the performance of variants of (3.2) in order to find an optimal strategy. In addition, the gain from applying further weights based on quantities such as element condition number should be investigated as it is suspected that this could be used to reduce convergence times for DDMs.

As all tests were carried out on a distributed memory parallel computer with SMP nodes (dual Xeons) an obvious optimisation for the graph partitioner is to first partition the mesh into the number of SMP nodes involved in the computation, and subsequently partition each of these partitions into two further partitions. These partition pairs are then assigned to the SMP nodes. This approach ensures that full advantage is taken of the fact that intra-nodal latency is much lower and communication bandwidth much higher than inter-nodal communication. Because this approach ensures that there is always a partition boundary

between two partitions on the same SMP node, it reduces the total cost of communication. The attraction of this approach is that it can easily be extended to any architecture which has a hierarchical communication pattern. This strategy could be used instead of, or in combination with, a mixed MPI/OpenMP approach, where OpenMP parallelises code running on a single SMP node.

Further optimisations for heterogeneous computational environments are possible. For example, a very interesting open source toolkit aimed at such resource-aware load-balancing has recently been released under the name of DRUM (Dynamic Resource Utilization Model) [113]. DRUM can be used by an application to provide a range of information about computational nodes being used by the application and the interconnect between these nodes. Tests range from *ping-pong* latency tests to the use of LINPACK [114] to run benchmarks to provide CPU power information. This information can be used by graph partitioners such as ParMetis to make optimal use of a given computational resource.

7.1.3 Data migration

As demonstrated in the previous chapter, the data migration module fulfils its function, and with little overhead when compared with the cost of other computation. So in the context of unstructured mesh based computation, such as that used in this work, it is unlikely to require further significant development. Nevertheless modelling today is rapidly moving more toward large scale coupled models. Research is currently underway toward coupling, for example, CFD with radiation transport [115], particle based modelling and discrete FEM, e.g. active tracer modelling for air pollution, sediment transport, individual based plankton modelling and coastal engineering. To satisfy all of the diverse requirements of these models for parallel computation and dynamic load-balancing the likelihood is that the open source toolkit Zoltan will be adopted [116, 117, 118].

7.2 Optimal shape approximation

A special case of optimal shape approximation is investigated in Chapter 5 - that of optimising the approximation of a height field such as bathymetry. The approach is attractive because of the flexibility in which the optimisation can be controlled. Examples of straightforward extentions are the inclusion of other field information (see the discussion of merging metrics in Section 2.4) and the simple control of errors (e.g. making interpolation error a function of bathymetry). The method was developed in a response to a pressing need to generate accurate surface meshes for ocean modelling using FEM and mesh optimisation.

However the method has two drawbacks. The first is that mesh optimisation is expensive in this case as typically a mesh gets compressed by two orders of magnitude (typically it is found that datasets over resolve areas such as abyssal planes while under resolving shelves). This cost is acceptable when using relatively small datasets such as GEBCO (0.5GBytes) but datasets such as sea bed surveys can involve terabytes of data. The second issue is that the method was developed to generate an optimal discrete representation of a domain that could be used with mesh optimisation methods throughout the course of a simulation. As discussed in Section 2.6, conservation is a critical issue to many applications (ocean modelling being one) thus methods by which an adaptive method might refer back to the origin geometry for new information are problematic. The issue is that method cannot be applied to general 3D geometry where there are similar needs. Variations of so called decimation methods [88] are common in the area of computer graphics for generating reduced representations of shapes that "look good" but these methods have in general a non-rigorous error norm which poses a problem for their use in quantitative science. However, more recent methods such as that developed in [119] (also developed for computer graphics) do have a rigorous error norm making it an ideal candidate for future application to shape approximation for FEM/FVM.

7.3 Conservative interpolation

When solving conservation equations, conservation of one or more quantities is often a fundamental requirement [120]. Thus it is important to make invariant certain integral quantities (e.g. mass, pollutant concentration, energy) throughout a time dependent simulation. The aim is to achieve conservation for linear and quadratic elements whilst using high order interpolations from an old mesh to a new one and simultaneously achieve bounded solutions e.g. non-negative densities and concentrations. These issues are analogous to conservative, high order accurate, bounded advection methods which require the introduction of non-linearity (Godunov's theorem) to simultaneously achieve all these aims and which must also be expected in mesh to mesh interpolation methods. Out of several approaches suggested in the literature, including [121, 122, 123], the two step approach proposed by Margolin et al. [123] is a likely starting point for achieving these aims. Also geometric conservation to preserve domain volume with free surfaces and other moving material interfaces will provide further conservation challenges. This may only be partially resolved with the increased accuracy associated with node movement and adaptive mesh refinement leaving the choice between sharp interfaces without conservation and slight smearing of the interfaces with conservation.

7.4 The next generation of mesh optimisation methods

It is worth considering the likely trend of mesh optimisation research for the next few years beyond that already described. Future work on mesh optimisation will aim to develop further advanced dynamic anisotropic mesh optimisation methods designed to be accurate, robust, and efficient. Current state-of-the-art mesh optimisation techniques are based on tetrahedral meshes - and research in this area is set to be active for many more years - although it is well known that tetrahedra perform poorly in comparison to hexahedral elements when used in FEM/FVM. Recent work on mesh optimisation and hex-dominant mesh generation (e.g. [124, 125, 126, 127, 128, 129, 130, 131]) has brought closer the prospect of a fully dynamic and robust hex-dominant mesh optimisa-

tion method.

The aim is to develop a mesh optimisation method which combines the accuracy of hexahedral elements with the geometric flexibility and ability to change resolution of tetrahedral elements. This will be achieved with a polyhedral mesh in which hexahedral elements are dominant, tetrahedral elements in regions constrained by complex geometry or similar, and pentahedra (wedges and pyramids) as transitional elements (thus the term *hex-dominant*).

7.4.1 Hex-dominant mesh optimisation

The challenge is to bring together mesh optimisation techniques developed for tetrahedral meshes (e.g. [3, 56, 8, 132]), parallel mesh optimisation methods (e.g. [53, 49, 32]), and developments in hex-dominant mesh generation (e.g. [124, 125, 126, 127, 128, 129, 130, 131]) to form a powerful hex-dominant mesh optimisation method.

Current state-of-the-art 3D unstructured mesh optimisation techniques are based on tetrahedral elements. However, when compared to hexahedral elements there are fundamental accuracy issues associated with using tetrahedral elements for 3D fluid modelling. Linear tetrahedral elements are well known to suffer from large false diffusion effects due to the absence of cross terms in the element shape functions. This reduction in accuracy not only occurs with simple linear elements but also with higher order elements such as used in spectral element methods. The benefits in terms of accuracy of using hexahedral elements are considerable as is the ability of legacy codes and other codes (e.g. multi-phase flow) that use hexahedral elements only, to exploit mesh optimisation techniques. Multi-phase models and ocean models often apply hexahedral elements since the use of other element types within their formulations are not possible, e.g. because of requirements to satisfy the LBB stability condition [104] (requires a different choice of element basis functions for the velocity and pressure to obtain numerical stability) and mass conservation. Similarly, some advection schemes also require the use of a hexahedral mesh. Indeed many experts in these fields believe it is not feasible to produce an accurate model with anything other than hexahedral elements. Furthermore, as the same solution accuracy can be

obtained with significantly fewer solution variables using a hexahedral mesh than with a tetrahedral mesh, much larger problems could be solved with a given computational resource.

A key aspect of this work will be the development and evaluation of a new family of objective functions to be used with mesh optimisation. These objective functions will need to describe in a consistent manner the quality of a given element type, size and shape with respect to a given metric tensor field derived from error norms. The objective function must naturally embody the local relative accuracy properties of different shapes and types of polyhedrons. This process will require extensive testing as the relative merits of different element types and shapes are difficult to quantify. Automatic differentiation techniques will be applied to the objective function (or simplifications of it) so that standard gradient methods may be used for parts of the mesh optimisation e.g. node positioning. The method for adapting the mesh will be based on several different techniques in the literature of mesh generation, in particular indirect hexahedral meshing methods (e.g. [124, 125, 126, 127, 128, 129, 130, 131]). The implementation of an indirect hex-dominant meshing method will be based on the current tetrahedral mesh optimisation method but applied with the newly developed objective function.

7.4.2 R-optimisation

Moving the mesh can be achieved with, for example, a Lagrangian method (although this is only a subset of the possibilities) to track materials or properties (in many cases exactly). However, when combined with mesh optimisation methods the mesh will be moved, coarsened, refined and have its element connectivity changed when the elements become distorted, as required.

All these methods, will also be combined with advanced moving mesh methods (e.g. [120]) to achieve levels of accuracy obtained only with Lagrangian methods, but with increased robustness, and conservative interpolation methods so that the models can have similar conservative properties to the governing equations.

As well as the necessity to preserve accuracy for free surface flow and moving material problems, the use of a moving mesh can substantially improve the accuracy of a simulation and even provide exact resolution of some advection processes. Mesh optimisation and mesh movement each have their own distinct advantages and disadvantages. For example, with mesh movement alone the number of nodes is kept constant; since the level of complex dynamic behaviour may vary throughout a simulation this is very unlikely to be near-optimal. In addition, it may be difficult to achieve in a straightforward manner large local variations in resolution, it is also possible that constraints in the topology of the mesh mean that inappropriately shaped or tangled meshes result. These problems would largely disappear if a component of mesh optimisation were available when mesh movement alone was found to be insufficient to cope with the solution dynamics. In contrast mesh optimisation alone may not be the most efficient way of performing adaptive simulations in (especially fluid based) situations, where regions which require enhanced resolution may well move with the flow in a Lagrangian manner. This could result in the excessive use of interpolation, especially when compared to a fully Lagrangian method which would typically require none. An additional resulting advantage of Lagrangian like methods is that in regions of intense dynamic activity small elements may well be present along with high velocities, and so any CFL based time step restriction present may be severe, an adaptive method which moves the mesh approximately with the flow would have the effect of reducing transport velocities, and hence also time step restrictions for these smaller elements, c.f. the properties of semi-Lagrangian methods. A combined method able to combine the robustness properties of mesh optimisation with the efficiency and accuracy properties of (Lagrangian) mesh movement is thus the ideal.

There are few examples of combined mesh optimisation/movement adaptive methods in the literature (and none for hex-dominant hybrid meshes). In [133] a 2D moving mesh method (based on variational elliptic mesh generation) is coupled with an h-method method which locally alters the mesh in order to achieve a desired error tolerance as well as improving poorly shaped elements. An alternate approach is used in [134] where a 2D mesh optimisation method is used which incorporates a moving mesh through repeated weighted Laplacian mesh smoothing. These references all demonstrate that the combined use of both mesh movement and mesh optimisation/refinement results in improvements

over either applied individually. In the same vein as the current mesh optimisation methods, new methods will be based on optimising an objective function which determines the need to fix the mesh to the moving properties (e.g. fluid). For example, a term may be added into the element shape/size quality function to represent the grid to grid interpolation errors (e.g. encouraging mesh movement perpendicular to solution gradients).

The optimisation of the mesh will involve a two stage approach. In the first stage, the node movement is optimised so that at the end of the time step the mesh is not overly distorted and the mesh has appropriate resolution and therefore accuracy is maximised. The nodes will be moved (relative to moving fields) in directions of low curvature (curvature will be normalised according to the element sizes) avoiding movement in directions of high curvature and maintaining element conformity to the boundaries of the domain. The second stage of the optimisation, optimises the mesh connectivity and node position, thus avoiding mesh tangling. This optimisation (previous section) does not have to be performed every time step and benefits (in terms of an improved starting point for the mesh optimisation) from the improved quality of the mesh from node movement from the first stage. In this second stage some of the element connectivity changes will be discouraged from occurring because of the additional interpolation errors resulting from mapping fields from the old mesh to the new adapted mesh.

The resulting method will have important applications in efficiently modelling of, for example, stratified fluid problems since the mesh will be able to approximately follow isopycnal surfaces, this holds the promise of yielding the largest benefit of current isopycnal models, namely minimised diapycnal mixing properties, along with increased robustness and advantages of z and sigma coordinate models. In addition, being aligned with isopycnals layers in stratified fluid applications is a property well known to be crucially important for long time-scale simulations. The incorporation of mesh alignment (or directionality information) is well known in variational structured mesh generation (e.g. [135]). Further, motivation from the well developed use of monitor functions (analogous to metric tensors here) in moving mesh methods (e.g. [136]) will be used as guides to additional developments.

7.4.3 Mesh adaptive multi-grid

Many mesh optimisation methods are based on hierarchical methods, e.g. [137, 132]. One of the advantages of this method is that the mesh hierarchy can be exploited for geometric multi-grid solver methods. When a hierarchical method is not employed one would still like to employ a multi-grid solver. One approach worthy of investigation is the use of a mesh optimisation method to generate a hierarchy of progressively coarser meshes. Standard grid-to-grid interpolation methods (e.g. [138]) would then be used to form the associated prolongation and restriction operators [139].

Two important trends in multi-grid are non-nested geometric multi-grid methods and algebraic multi-grid methods. Typically, in geometric multi-grid coarse representations of the fine mesh are obtained and a FE assembly is performed on each coarse mesh. The FE on the coarsest level is then solved and the solution interpolated onto the next mesh. This provides the starting point for the next application of the smoother. In contrast, algebraic multi-grid uses aggregation to lump terms in the matrix, recursively reducing its rank, until it can be solved efficiently. The latter method has proved popular because it can be applied as a black box algorithm and aggregation is algorithmically less complex than generating coarse meshes. However, the method can lead to spurious coupling between terms in the matrix. An alternative algorithm uses geometric multi-grid which defines the restriction operator as the interpolation operator (based on the finite element shape functions) and similarly define the prolongation operators (see Adams [139]). This can then be used with standard algebraic multi-grid methods. In order to generate the hierarchy of meshes Adams [139] calculates maximum independent sets (MIS) to generate smaller subsets of elements and then re-triangulate using Delaunay triangulation. This reduces the impact of spurious coupling between terms but is not optimal, the Delaunay re-triangulation step is expensive and naturally only tetrahedra are employed. An alternative approach is to create a specialised form of the mesh optimisation method for rapidly generating a hierarchy of coarser meshes from a given fine mesh. Recent edge coarsening methods such as that proposed by Ollivier-Gooch [46] are particularly interesting. This approach will maximise coupling between terms in the matrix when the reduction/prolongation operators are applied. These multi-grid meshes can be generated

to reflect the solution structure or the conditioning of the matrix and provides extreme flexibility in their construction. Parallelisation of this method is greatly simplified by minimising coarsening across sub-domain boundaries.

7.4.4 Error norms

The overall optimisation method will adapt and optimise hex-dominant unstructured meshes with respect to a metric tensor embodying all the solution errors. The metric tensors will either be based on the Hessian of the solution (e.g. [10, 9, 11]) or a posteriori error measures [12, 13, 14, 140]).

The aim of goal based error measures is to take the emphasis off the numerical analyst in designing error norms and put it in the domain of the engineer or physicist who has a firm understanding of the problem at hand. To do this one would typically want to extract a particular quantity from the numerical simulation, for example an integral quantity such as vorticity. These methods are particularly important for problems with a number of solution variables where it is typically unclear what priority to put on resolving each of the fields. In addition, sensitivity information may also be used to provide a bound for the error in the required quantity, which can be invaluable to any model. An example of an application area of this type of technique is in adapting a mesh to optimise a quantity of interest in a fluid simulation such as the drag or lift past an aerofoil. The tendency of an adapting mesh to refine right down to the scale able to resolve molecular viscosity in large scale turbulent flows (e.g. in an ocean) can be reduced, as the adaption technique can be guided by the measure of what is deemed to be important in the physics of the model.

7.4.5 Parallel computing

An important component of any new method needs to be the inclusion of parallel computational support. This means that at each stage of the method development the scalability of the overall algorithm must be maintained as to avoid problems when parallelising the method.

Parallel mesh optimisation is currently performed by exploiting a domain decomposition method in which the domain is partitioned into sub-domains and each processor is responsible for optimising a single sub-domain, excluding the overlapping region [53]. The main difficulty is the problem of adapting elements that are shared between domains. In order to maintain a consistent mesh these shared elements need to be updated in an identical fashion across all domains. Parallel mesh optimisation is achieved by adapting all the elements in each sub-domain independently while locking the overlapping elements so that they are left unchanged. A graph repartitioning is required to re-balance the number of mesh nodes (processor load) per domain. The error metric is used to give a predicted number of nodes in regions that have not been optimised. The predicted value is used to apply weights to the mesh nodes to maximise the accuracy of the load-balance. In addition to this, weights are applied to the graph edges such that edges associated with poor elements have high weights and so are less likely to be split by a partition. This choice of edge weight, has the effect of perturbing partitions away from elements that require optimisation while still load-balancing. Mesh optimisation is then reapplied to the mesh to address these sub-standard elements. For repartitioning the parallel multilevel graph partitioner ParMetis [66] is used. While this approach is effective it could be greatly improved with the introduction of an efficient moving mesh method. Moving mesh methods are extremely attractive as they do not adversely affect load-balance and as only nodal positions are updated it is straight forward to move halo nodes in parallel [4]. This would greatly decrease the number of repartitions, and thus the volume of data migration, that would be required.

Bibliography

[1] In *International Workshop on Unstructured Grid Numerical Modelling of Coastal, Shelf and Ocean Flows, Delft 2003*, 2004.

[2] C. C. Pain, M. D. Piggott, A. J. H. Goddard, F. Fang, G. J. Gorman, D. P. Marshall, M. D. Eaton, P. W. Power, and C. R. E. de Oliveira. Three-dimensional unstructured mesh ocean modelling. *Ocean modelling, special issue on unstructured mesh ocean modelling*, To appear in 2005.

[3] C. C. Pain, A. P. Umpleby, C. R. E. de Oliveira, and A. J. H. Goddard. Tetrahedral mesh optimisation and adaptivity for steady-state and transient finite element calculations. *Comput. Methods Appl. Mech. Engrg.*, 190:3771–3796, 2001.

[4] M. D. Piggott, C. C. Pain, G. J. Gorman, P. W. Power, and A. J. H. Goddard. *h*, *r* and *hr* adaptivity in ocean modelling. *Ocean modelling, special issue on unstructured mesh ocean modelling*, To appear in 2005.

[5] P. W. Power, C. C. Pain, M. D. Piggott, G. J. Gorman, D. P. Marshall, and A. J. H. Goddard. Error norms for ocean modelling utilizing adjoint sensitivity techniques. *Ocean modelling, special issue on unstructured mesh ocean modelling*, Under revision, 2005.

[6] P. L. George. *Automatic mesh generation: Application to finite element methods*. Wiley, 1991.

[7] O. C. Zienkiewicz and R. L. Taylor. *The finite element method*, volume 1. Butterworth-Heinemann, 5th edition, 2002.

[8] L. A. Freitag and C. Ollivier-Gooch. Tetrahedral mesh improvement using swapping and smoothing. *Int. J. Num. Methods Engrg.*, 40:3979–4002, 1997.

[9] R. B. Simpson. Anisotropic mesh transformations and optimal error control. *Applied Numerical Mathematics: Transactions of IMACS*, 14(1–3):183–198, 1994.

[10] E. F. D'Azevedo and R. B. Simpson. On optimal triangular meshes for minimizing the gradient error. *Numer. Math.*, 59:321–348, 1991.

[11] J. R. Shewchuk. What is a good linear element? Interpolation, Conditioning, and Quality Measures. In *Eleventh International Meshing Roundtable*, 2002.

[12] D. W. Kelly, R. J. Mills, and J. A. Rezes. A posteriori estimates of the solution error caused by discretization in the finite element methods. *International Journal for Numerical Methods in Engineering*, pages 1921–1939, 1987.

[13] M. Ainsworth, J. Z. Zhu, A. W. Craig, and O. C. Zienkiewicz. Analysis of the Zienkewicz-Zhu a-posteriori error estimator in the finite element method. *International Journal for Numerical Methods in Engineering*, pages 2161–2174, 1989.

[14] O. C. Zienkiewicz and J. Z. Zhu. The superconvergant patch recovery and posteriori error estimates. Part 1: The recovery technique. *International Journal for Numerical Methods in Engineering*, pages 1331–1364, 1992.

[15] J. Peraire, M. Vahdati, K. Morgan, and O. C. Zienkiewicz. Adaptive remeshing for compressible flow computations. *J. Comput. Phys.*, 72:449–466, 1987.

[16] J. Peraire, K. Morgan, M. Vahdati, and O. C. Zienkiewicz. Finite element Euler computations in 3-d. *Internat. J. Num. Meth. Eng.*, 26:2135–2159, 1988.

[17] N. P. Weatherill, P. R. Eiseman, J. Hauss, and J. F. Thompson. *Numerical grid generation in computational fluid dynamics and related fields*. Pineridge Press, Swansea, 1994.

[18] P. L. George and H. Borouchaki. *Delaunay triangulation and meshing: Application to finite elements*. Editions Hermes, 1998.

[19] X. Xu, C. C. Pain, A. J. H. Goddard, and C. R. E. de Oliveira. An automatic adaptive meshing technique for Delaunay triangulations. *Computer methods in applied mechanics and engineering*, 161:297–303, 1997.

[20] J. F. Thompson, B. K. Soni, and N. P. Weatherill, editors. *Handbook of grid generation*. CRC Press, 1999.

[21] R. E. Bank and A. H. Sherman. The use of adaptive grid refinement for badly behaved elliptic partial differential equations. *Advances in Computer Methods for Partial Differential Equations*, 3:33–39, 1979.

[22] R. E. Bank and A. H. Sherman. An adaptive, multigrid method for elliptic boundary value problems. *Computing*, 26:91–105, 1981.

[23] R. E. Bank, A. H. Sherman, and A. Weiser. Refinement algorithm and data structures for regular local mesh refinement. *Scientific Computing*, 44:3–17, 1983.

[24] M. Rivara. Mesh refinement processes based on the generalized bisection of simplices. *SIAM Journal on Numerical Analysis*, 21:604–613, 1984.

[25] M. Rivara. Algorithms for refining triangular grids suitable for adaptive and multigrid techniques. *International Journal for Numerical Methods in Engineering*, 20:745–756, 1984.

[26] R Löhner and J. D. Baum. Adaptive h-refinement on 3d unstructured grids for shock hydro-dynamics. *Int. J. Num. Meth. Fluids*, 14:1407–1419, 1992.

[27] M. E. G. Ong. Uniform refinement of tetrahedron. *SIAM J. Sci. Comput.*, 15(5), 1994.

[28] W. Speares and M. Berzins. A 3-d unstructured mesh adaption algorithm for time-dependant shock dominated problems. *Int. J. Num. Meth. Fluids*, 25(1):81–104, 1997.

[29] L. Oliker, R. Biswas, and R. C. Strawn. Parallel implementation of an adaptive scheme for 3D unstructured grids on the SP2. In *IRREGULAR '96: Proceedings of the Third International Workshop on Parallel Algorithms for Irregularly Structured Problems*, pages 35–47, 1996.

[30] P. M. Selwood, M. Berzins, and P. M. Dew. 3d parallel mesh adaptivity: Data-structures and algorithms. *SIAM: Scientific Computing*, 1997.

[31] N. Touheed, P. Selwood, P. K. Jimack, and M. Berzins. A comparisons of some dynamic load-balancing algorithms for a parallel adaptive flow solver. In B. H. V. Topping, editor, *Parallel and Distributed Processing for Computational Mechanics II*. Saxe-Coburg Publications, 1998.

[32] P. K. Jimack. Techniques for parallel adaptivity. In B. H. V. Topping, editor, *Parallel and Distributed Processing for Computational Mechanics II*. Saxe-Coburg Publications, 1998.

[33] P. M. Selwood and M. Berzins. Parallel unstructured tetrahedral mesh adaptation: algorithms, implementation and scalability. *Concurrency: Practice and Experience*, 11(14):863–884, 1999.

[34] L. Oliker, R. Biswas, and H. N. Gabow. Parallel tetrahedral mesh adaptation with dynamic load balancing. *Parallel Computing*, 26(12):1583–1608, 2000.

[35] E. B. l'Isle and P. L. George. *Optimization of tetrahedral meshes*, pages 97–127. Springer, Berlin, 1995.

[36] B. Joe. Three-dimensional triangulations from local transformations. *SIAM J. Sci. Stat. Comput.*, 10(4):718–741, 1989.

[37] B. Joe. Construction of three-dimensional Delaunay triangulations using local transformations. *Computer Aided Geom. Design*, 8:123–142, 1991.

[38] H. Edelsbrunner and N. R. Shah. Incremental topological flipping works for regular triangulations. In *SCG '92: Proceedings of the eighth annual symposium on Computational geometry*, pages 43–52, 1992.

[39] B. Joe. Construction of three-dimensional improved-quality triangulations using local transformations. *SIAM J. Sci. Comput.*, 16:1292–1307, 1995.

[40] D. A. Field. Laplacian smoothing and Delaunay triangulations. *Communications in Applied Numerical Methods*, 4:709–712, 1988.

[41] V. N. Parthasarathy and S. Kodiyalam. A constrained optimization approach to finite element mesh smoothing. *Finite Elem. Anal. Des.*, 9(4):309–320, 1991.

[42] S. A. Canann, M. B. Stephenson, and T. Blacker. Optismoothing: An optimization-driven approach to mesh smoothing. *Finite Elements in Analysis and Design*, 13:185–190, 1993.

[43] N. Amenta, M. Bern, and D. Eppstein. Optimal point placement for mesh smoothing. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1997.

[44] P. M. Knupp. Achieving finite element mesh quality via optimization of the Jacobian matrix norm and associated quantities. Part II - A framework for volume mesh optimisation and the condition number of the [j]acobian matrix. *Int. J. Numer. Meth. Engng*, 48:1165–1185, 2000.

[45] G. C. Buscaglia and E. A. Dari. Anisotropic mesh optimization and its applications in adaptivity. *Int. J. Num. Methods Engrg.*, 40:4119–4136, 1997.

[46] C. Ollivier-Gooch. Coarsening unstructured meshes by edge contraction. *Int. J. Num. Meth. Eng.*, 57(3):391–414, 2003.

[47] M. T. Jones and P. E. Plassmann. Parallel algorithms for adaptive mesh refinement. *SIAM J. Sci. Comp.*, 18, 1997.

[48] L. A. Freitag, M. T. Jones, and P. E. Plassmann. A parallel algorithm for mesh smoothing. *SIAM J. Sci. Comp.*, 20(6), 1999.

[49] L. Freitag, M. Jones, and P. Plassmann. *The scalability of mesh improvement algorithms*, pages 185–212. Springer-Verlag, 1998.

[50] X. Li, J. F. Remacle, N. Chevaugeon, and M. S. Shephard. Anisotropic mesh gradation control. In *13th International Meshing Roundtable*, pages 401–412, 2004.

[51] R. Courant, K. O. Friedrichs, and H. Lewy. Uber die partiellen differenzgleichungen der mathematischen physik. *Mathematische Annalen*, 100:32–74, 1928.

[52] J. E. Flaherty, R. M. Loy, M. S. Shephard, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz. Predictive load balancing for parallel adaptive finite element computation. In H. R. Arabnia, editor, *Proc. PDPTA*, pages 460–469, 1997.

[53] G. J. Gorman, C. C. Pain, C. R. E. de Oliveira, A. P. Umpleby, and A. J. H. Goddard. Parallel unstructured mesh optimisation for 3d radiation transport and fluids modelling. In *International Conference on Supercomputing in Nuclear Applications*, 2003.

[54] F. Fang, C. C. Pain, M. D. Piggott, G. J. Gorman, A. J. H. Goddard, G. M. Copeland, and I. Gejadze. An adaptive mesh adjoint data assimilation method applied to free surface flow problems. *Int. J. Numer. Meth. Fluids. In press*, 2004.

[55] G. J. Gorman, M. D. Piggott, C. C. Pain, C. R. E. de Oliveira, A. P. Umpleby, and A. J. H. Goddard. Optimisation based bathymetry approximation through constrained unstructured mesh adaptivity. *Ocean Modelling*, 12(3-4):436–452, 2006.

[56] L. A. Freitag and P. M. Knupp. Tetrahedral mesh improvement via optimization of the element condition number. *International Journal for Numerical Methods in Engineering*, 53(6):1377–1391, 2002.

[57] L. A. Freitag, M. T. Jones, and P. E. Plassmann. An efficient parallel algorithm for mesh smoothing. *Int. Meshing Roundtable*, 1995.

[58] B. Smith, P. Bjørstad, and W. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996.

[59] B. W. Kernighgan and S. Lin. An efficient procedure for partitioning graphs. *Bell Sys. Tech. J.*, 49, 1970.

[60] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Proc. Supercomputing '95*, 1995. Formerly, Technical Report SAND93-1301 (1993).

[61] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. on Sci. Comput.*, 20(1):359–392, 1998.

[62] C. Walshaw, M. Cross, and M. Everett. Parallel dynamic graph partitioning for adaptive unstructured meshes. *Journal of Parallel and distributed Computing*, 47:102–108, 1997.

[63] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoret. Comput. Sci.*, 1, 1976.

[64] C. C. Pain, C. R. E. Oliveira, and A. J. H. Goddard. A neural network graph partitioning procedure for grid-based domain decomposition. *International journal for numerical methods in engineering*, pages 593–613, 1999.

[65] A. J. Soper, C. Walshaw, and M. Cross. A combined evolutionary search and multilevel optimisation approach to graph partitioning. Technical report, Univ. Greenwich, London SE10 9LS, UK, April 2000.

[66] G. Karypis, K. Schloegel, and V. Kumar. Parmetis: Parallel graph partitioning and sparse matrix ordering library. http://www-users.cs.umn.edu/ karypis/metis/parmetis/, 2004.

[67] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. In *Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*, page 35. ACM Press, 1996.

[68] K. Schloegel, G. Karypis, and V. Kumar. Multilevel diffusion schemes for repartitioning of adaptive meshes. *Journal of Parallel and Distributed Computing*, 47(2):109–124, 1997.

[69] K. Schloegel, G. Karypis, and V. Kumar. A unified algorithm for load-balancing adaptive scientific simulations. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, page 59. IEEE Computer Society, 2000.

[70] B. Hendrickson. Graph partitioning and parallel solvers: Has the emperor no clothes? (extended abstract). In *Workshop on Parallel Algorithms for Irregularly Structured Problems*, pages 218–225, 1998.

[71] K. Schloegel, G. Karypis, and V. Kumar. Wavefront diffusion and LMSR: Algorithms for dynamic repartitioning of adaptive meshes. *IEEE Transactions on Parallel and Distributed Systems*, 12(5), 2001.

[72] C. Ozturan. *Distributed Environment and load balancing for adaptive unstructured meshes*. PhD thesis, Rensselaer Polytechnic Institute, Troy, New York, 1995.

[73] SGI. Standard template library programmer's guide. http://www.sgi.com/tech/stl/, 1994.

[74] C. C Pain. Brief description and capabilities of the general cfd code fluidity. Technical report, Imperial College, 2000.

[75] C. R. E. de Oliveira. An arbitrary geometry finite element method for multigroup neutron transport with anisotropic scattering. *Prog. Nucl. Energy*, 18:227, 1986.

[76] S. Byna, W. Gropp, X.-H. Sun, and R. Thakur. Improving the performance of mpi derived datatypes by optimizing memory-access cost. In *IEEE International Conference on Cluster Computing*, 2003.

[77] O. C. Zienkiewicz and R. L. Taylor. *The finite element method*, volume 2. McGraw-Hill Book Company, 4th edition, 1991.

[78] A. Adcroft and D. Marshall. How slippery are piecewise-constant coastlines in numerical ocean models? *Tellus A*, 50:95–108, 1998.

[79] T. M. Özgökmen, E. P. Chassignet, and A. M. Paiva. Impact of wind forcing, bottom topography, and inertia on midlatitude jet separation in a quasigeostropic model. *J. Phys. Oceangr.*, 27:2460–2476, 1997.

[80] M. E. Stern. Separation of a density current from the bottom of a continental shelf. *J. Phys. Oceanogr.*, 28:2040–2049, 1998.

[81] C. E. Tansley and D. P. Marshall. On the influence of bottom topography and the deep western boundary current on Gulf Stream separation. *J. Mar. Res.*, 58:297–325, 2000.

[82] D. R. Munday and D. P. Marshall. On the separation of a barotropic western boundary current from a cape. *J. Phys. Oceanogr.*, In Press 2005.

[83] M. J. Roberts and R. A. Wood. Topographic sensitivity studies with a Bryan-Cox-Type ocean model. *J. Phys. Oceanogr.*, 27(5):823–836, 1997.

[84] R. L. Haney. On the pressure gradient force over steep topography in sigma coordinate models. *J. Phys. Oceanogr.*, 21:610–619, 1991.

[85] R. F. Henry and R. A. Walters. Geometrically based, automatic generator for irregular triangular networks. *Commun. Num. Math. Eng.*, 9(7):555–566, 1993.

[86] J. R. Shewchuk. Triangle. http://www-2.cs.cmu.edu/∼quake/triangle.html, 2003.

[87] S. Legrand, V. Legat, and E. Deleersnijder. Delaunay mesh generation for an unstructured-grid ocean general circulation model. *Ocean Modelling*, 2:17–28, 2000.

[88] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of triangle meshes. *Computer Graphics*, 26(2):65–70, 1992.

[89] IOC, IHO, and BODC. Centenary edition of the GEBCO digital atlas. Published on CD-ROM on behalf of the Intergovernmental Oceanographic Commission and the International Hydrographic Organization as part of the General Bathymetric Chart of the Oceans, 2003.

[90] B. K. Karamete, R. V. Garimella, and M. S. Shephard. Recovery of an arbitrary edge on an existing surface mesh using local mesh modifications. *Int. J. Numer. Methods Eng.*, 50:1389–1409, 2001.

[91] E. A. Soluri and V. A. Woodson. World Vector Shoreline. *International Hydrographic Review*, LXVII(1), 1990.

[92] J. Snyder. Map projections — A working manual. *U.S. Geological Survey Professional Paper 1395, U.S. Government Printing Office, Washington, D.C.*, pages 154–163, 1987.

[93] J. Dengg, A. Beckmann, and R. Gerdes. The Gulf Stream separation problem. In W. Kraus, editor, *The warm water sphere of the North Atlantic Ocean*, pages 253–289. Gebruder Borntrager, 1996.

[94] J. Dengg. The problem of Gulf Stream separation: A baratropic approach. *J. Phys. Oceanogr.*, 23:2182–2200, 1993.

[95] W. R. Holland. Baroclinic and topographic influences on the transport in western boundary currents. *Geophys. Fluid Dyn.*, 4:187–210, 1973.

[96] A. Beckmann, C. W. Böning, C. Köbele, and J. Willebrand. Effects of increased horizontal resolution in simulations of the North Atlantic Ocean. *J. Phys. Oceanogr.*, 24:326–344, 1995.

[97] Y. Chao, A. Gangopadhyay, F. O. Bryan, and W. R. Holland. Modeling the Gulf Stream system: how far from reality? *Geophys. Res. Lett.*, 23:3155–3158, 1996.

[98] R. D. Smith, M. E. Maltrub, F. O. Bryan, and M. W. Hecht. Numerical simulation of the North Atlantic Ocean at $1/10^{\circ}$. *J. Phys. Oceanogr.*, 30:1532–1561, 2000.

[99] L. Jiang and R. W. Garwood Jr. Effects of topographic steering and ambient stratification on overflows on continental slopes: A model study. *J. Geophys. Res.*, 103(C3):5459–5476, 1998.

[100] R. Gerdes. A primitive equation ocean circulation model using a general vertical coordinate transformation 2. Application to an overflow problem. *J. Geophys. Res.*, 98(C8):14703–14726, 1993.

[101] T. M. Özgökmen. DOME. http://www.rsmas.miami.edu/personal/tamay/DOME/dome.html, 2001.

[102] J. F. Price and M. Baringer. Outflows and deep water production by marginal seas. *J. Phys. Oceanogr.*, 33:161–200, 1994.

[103] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48, 1998.

[104] I. Babuska. Error-bounds for finite element method. *Numer. Math.*, 16:322–333, 1971.

[105] S. W. Armfield. Finite difference solutions of the Navier-Stokes equations on staggered and non-staggered grids. *Comput. Fluids*, 20:1–88, 1991.

[106] C. Guiang, K. Milfeld, A. Purkayastha, and J. Boisseau. Memory performance on dual-processor nodes: Comparison of Intel Xeon and AMD opteron memory subsystem architectures. In *Proceedings of the ClusterWorld Conference and Expo, San Jose, CA*, 2003.

[107] R. Hide. On the dynamics of rotating fluids and related topics in geophysical fluid dynamics. *Bulletin of the American Meteorological Socity*, 47(11), 1966.

[108] C. C. Pain, M. D. Piggott, A. J. H. Goddard, F. Fang, G. J. Gorman, D. P. Marshall, M. D. Eaton, P. W. Power, and C. R. E. de Oliveira. Three-dimensional unstructured mesh ocean modelling. *Ocean modelling, special issue on unstructured mesh ocean modelling. To appear*, 2005.

[109] D. H. Bailey. Twelve ways to fool the masses when giving performance results on parallel computers. *Supercomputing Review*, 4(8):54–55, 1991.

[110] J. E. Flaherty, R. M. Loy, M. S. Shephard, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz. Predictive load balancing for parallel adaptive finite element computation. In H. R. Arabnia, editor, *Proc. PDPTA '97*, volume I, pages 460–469, 1997.

[111] L. A. Freitag and C. Ollivier-Gooch. A comparison of tetrahedral mesh improvement techniques. In 5^{th} *International Meshing Roundtable*, 1996.

[112] L. A. Freitag and C. Ollivier-Gooch. A cost/benefit analysis of simplicial mesh improvement techniques as measured by solution efficiency. *International Journal of Computational Geometry and Applications*, 10(4):361–382, 2000.

[113] J. Faik, L. G. Gervasio, J. E. Flaherty, J. Chang, J. D. Teresco, E. G. Boman, and K. D. Devine. A model for resource-aware load balancing on heterogeneous

clusters. Technical Report CS-03-03, Williams College Department of Computer Science, 2003.

[114] J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart. Linpack users guide. SIAM, Philadelphia, PA, 1979.

[115] C. C. Pain, C. R. E. de Oliveira, M. D. Eaton, and A. J. H. Goddard. Insights into the JCO criticality incident using the coupled radiation/ multiphase code FETCH. In *PHYSOR 2002, Seoul, Korea*, 2002.

[116] K. D. Devine, E. G. Boman, R. T. Heaphy, B. A. Hendrickson, J. D. Teresco, J. Faik, J. E. Flaherty, and L. G. Gervasioc. New challenges in dynamic load balancing. *To appear in Applied Numerical Mathematics*, 2004.

[117] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan. Zoltan data management services for parallel dynamic applications. *Comput. Sci. Engrg.*, 4(2), 2002.

[118] K. Devine, B. Hendrickson, E. Boman, M. St. John, and C. Vaughan. Zoltan: A dynamic load balancing library for parallel applications; users guide. Technical Report SAND99-1377, Sandia National Laboratories, Albuquerque, NM, 1999.

[119] D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. *To appear, ACM SIGGRAPH*, 2004.

[120] C. J. Budd and M. D. Piggott. Geometric integration and its applications. *Handbook of Numerical Analysis*, XI:35–139, 2003.

[121] H. Tang and T. Tang. Adaptive mesh methods for one and two-dimensional hyperbolic conservation laws. *SIAM J. Numer. Anal.*, 41(4), 2003.

[122] J. Grandy. Conservative remapping and region overlays by intersecting arbitrary polyhedra. *J. Comput. Phys.*, 148, 1999.

[123] L. G. Margolin and M. Shashkov. Second-order sign-preserving conservative interpolation (remapping) on general grids. *J. Comp. Phys.*, 184:266–298, 2003.

[124] M. Brewer, L. F. Diachin, P. Knupp, T. Leurent, and D. Malander. The mesquite mesh quality improvement toolkit. In *12th International Meshing Roundtable*, 2003.

[125] S. Yamakawa and K. Shimada. Increasing the number and volume of hexahedral and prism elements in a hex-dominant mesh by topological transformations. In *12th International Meshing Roundtable*, 2003.

[126] T. J. Tautges and S. E. Knoop. Topological modification of hexahedral meshes using atomic dual-based operations. In *12th International Meshing Roundtable*, 2003.

[127] S. Paoletti. Polyhedral mesh optimization using the interpolation tensor. In *11th International Meshing Roundtable*, 2002.

[128] K.-F. Tchon, J. Dompierre, and R. Camarero. Conformal refinement and all-quadralateral and all-hexahedral meshes according to an anisotropic metric. In *11th International Meshing Roundtable*, 2002.

[129] M. Bern and D. Eppstein. Flipping cubical meshes. In *10th International Meshing Roundtable*, 2001.

[130] S. J. Owen and S. Saigal. H-morph: an indirect approach to advancing front hex meshing. *Int. J. Numer. Meth. Engng*, 49, 2000.

[131] S. Meshkat and D. Talmor. Generating a mixed mesh of hexahedra, pentahedra and tetrahedra from an underlying tetrahedral mesh. *Int. J. Numer. Meth. Engng*, 49(1):17–30, 2000.

[132] P. K. Jimack and R. Mahmood. A multilevel approach for obtaining locally optimal finite element meshes. *Adv. Eng. Softw.*, 33(7-10):403–415, 2002.

[133] J. Lang, W. Cao, W. Huang, and R. D. Russell. A two-dimensional moving finite element method with local refinement based on a posteriori error estimates. *Appl. Numer. Meth.*, 46:75–94, 2003.

[134] J. Dompierre, M. G. Vallet, Y. Bourgault, M. Fortin, and W.G. Habashi. Anisotropic mesh adaption: towards user-independent, mesh-independent and solver-independent cfd. *Int. J. Numer. Methods Fluids*, 39:675–702, 2002.

[135] J. U. Brackbill. An adaptive grid with directional control. *J. Comput. Phys.*, 108:38–50, 1993.

[136] W. Cao, W. Huang, and R. D. Russell. A study of monitor functions for two dimensional adaptive mesh generation. *SIAM J. Sci. Comp.*, 20, 1999.

[137] M. Berzins, P. K. Jimack, and M. Walkley. Mesh quality and moving meshes for 2d and 3d unstructured mesh flow solvers. In 31st VKI CFD Lecture Series, 2000.

[138] R. Löhner. Robust, vectorized search algorithms for interpolation on unstructured grids. *Journal of Computational Physics*, 118:380, 1995.

[139] M. Adams. Evaluation of three unstructured multigrid methods on 3d finite element problems in solid mechanics. *Int. J. Numer. Meth. Engng*, 55(5), 2002.

[140] I. M. Babuska and R. Rodrigues. The problem of the selection of an a posteriori error indicator based on smoothing techniques. *International Journal for Numerical Methods in Engineering*, pages 539–567, 1993.

Appendix A

Metric tensor

The metric tensor $\boldsymbol{M}$, also called a Riemannian metric, is a symmetric and positive definite tensor which may be used as a function that calculates the distance between any two points to be computed in a given space, Ω. The components of $\boldsymbol{M}$, g_{ij}, can be viewed as multiplication factors which must be placed in front of the differential displacements dx_i in a generalised Pythagorean theorem,

$$ds^2 = \boldsymbol{dx}^T \boldsymbol{M} \boldsymbol{dx}. \tag{A.1}$$

In Euclidean space, $g_{ij} = \delta_{ij}$ (where δ_{ij} is the Kronecker delta defined by, $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ otherwise), this reduces to the more familiar form of the Pythagorean theorem,

$$ds^2 = dx_1^2 + dx_2^2 + \cdots .$$

The metric itself is coordinate independent but it can be expressed in terms of components that depend on the choice of coordinate system. To find the components in some

coordinate system, supply the basis as arguments to the metric function,

$$g_{\alpha\beta} =< \boldsymbol{e}_\alpha, \boldsymbol{e}_\beta > .$$

In Cartesian coordinates, $< \boldsymbol{e}_\alpha, \boldsymbol{e}_\beta >= \delta_{\alpha\beta}$ and therefore,

$$\begin{pmatrix} g_{xx} & g_{xy} & \cdots \\ g_{yx} & g_{yy} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots \\ 0 & 1 & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

Take for example metric components in plane polar coordinates:

$$\begin{aligned} x = &\quad r\cos(\theta), \\ y = &\quad r\sin(\theta), \end{aligned}$$

where θ is the angle between the vector and $\boldsymbol{e}_x$. The basis vectors for plane polar coordinates are:

$$\begin{aligned} \boldsymbol{e}_r = &\quad \boldsymbol{e}_x \left(\tfrac{\partial x}{\partial r}\right) + \boldsymbol{e}_y \tfrac{\partial x}{\partial r}, \\ = &\quad \boldsymbol{e}_x \cos(\theta) + \boldsymbol{e}_y \sin(\theta), \end{aligned}$$

$$\begin{aligned} \boldsymbol{e_\theta} = &\quad \boldsymbol{e}_x \left(\tfrac{\partial x}{\partial \theta}\right) + \boldsymbol{e}_y \tfrac{\partial x}{\partial \theta}, \\ = &\quad -\boldsymbol{e}_x r\sin(\theta) + \boldsymbol{e}_y r\cos(\theta). \end{aligned}$$

Therefore the components of the metric associated with the new coordinates are:

$$\begin{aligned} g_{rr} = &\quad < \boldsymbol{e}_r, \boldsymbol{e}_r >= \quad 1, \\ g_{r\theta} = &\quad < \boldsymbol{e}_r, \boldsymbol{e}_\theta >= \quad 0, \\ g_{\theta\theta} = &\quad < \boldsymbol{e}_\theta, \boldsymbol{e}_\theta >= \quad r^2. \end{aligned}$$

Expressing the metric tensor $\boldsymbol{M}$ in terms of its eigen decomposition,

$$\boldsymbol{M} = \boldsymbol{V}^T \boldsymbol{\Lambda} \boldsymbol{V},$$

where $\boldsymbol{V}$ contains the eigenvectors and $\boldsymbol{\Lambda} = diag(\lambda_i), i = 1, 2, ...$ is the diagonal matrix of eigenvalues, a useful physical interpretation of the metric tensor is evident. The metric tensor can be seen as a description of how space is stretched or compressed along the direction of the eigenvectors. Specifically, it can be expressed in terms of its deformation matrix, $\boldsymbol{M} = \boldsymbol{F}^T \boldsymbol{F}$, where $\boldsymbol{F} = \boldsymbol{\Lambda}^{\frac{1}{2}} \boldsymbol{V}$. Thus, a vector $\boldsymbol{v}$ can be mapped from the reference space to the space Ω using,

$$\boldsymbol{v'} = \boldsymbol{F}^T \boldsymbol{v}.$$

A similar transformation exists for transforming a metric tensor from space Ω_1, $\boldsymbol{M}_1$, to another space Ω_2, $\boldsymbol{M}_2$:

$$\boldsymbol{M'} = \boldsymbol{F}_2^{-T} \boldsymbol{M}_1 \boldsymbol{F}_2^{-1}.$$

Also note,

$$\boldsymbol{F}_1^{-T} \boldsymbol{M}_1 \boldsymbol{F}_1^{-1} = \boldsymbol{I}.$$